AFRL-IF-RS-TR-2005-269
Final Technical Report
July 2005

EAGLE HATS MINI-TECHNOLOGY INTEGRATION EXPERIMENT (TIE)

University of Massachusetts at Amherst

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-269 has been reviewed and is approved for publication

APPROVED: /s/

WILLIAM E. RZEPKA
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JULY 2005	Final Sep 01 – Nov 04

4. TITLE AND SUBTITLE
EAGLE HATS MINI-TECHNOLOGY INTEGRATION EXPERIMENT (TIE)

5. FUNDING NUMBERS
C - F30602-01-2-0580
PE - 31011G
PR - EELD
TA - 01
WU - 07

6. AUTHOR(S)
Paul R. Cohen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Massachusetts
Experimental Knowledge Systems Laboratory
Computer Science Department
Amherst Massachusetts 01003

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFED
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-269

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: William E. Rzepka/IFED/(315) 330-2762/ William.Rzepka@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

Hats is a virtual world in which agents (called hats) move around, go to meetings, acquire capabilities, do business, and, for a small subpopulation of agents, do harm. Agents move on a two-dimensional board that has only two kinds of locations: Beacons are high-value places that terrorist agents would like to destroy, other locations have low value. All beacons have a set of attributes, or vulnerabilities, corresponding to the capabilities agents carry. To destroy a beacon, a terrorist task force must be in possession of a set of capabilities that match a beacon's vulnerabilities, as a key matches a lock. In general, these sets of capabilities are not unique to terrorists, so one cannot identify a terrorist task force from its constituent capabilities alone. Hats serves as an unclassified proxy for real Intelligence analysis.

14. SUBJECT TERMS
Simulator, Bayesian Blackboard, Group Finding Algorithm

15. NUMBER OF PAGES
50

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

Executive Summary

The Experimental Knowledge Systems Laboratory (EKSL) at then University of Massachusetts, Amherst supported the Hats mini-TIE of the AFRL EAGLE project. The goal of the mini-TIE was to develop and publicize Hats as a strong and unclassified contender for research into algorithms, tools and techniques for intelligence analysis. Our chief accomplishments were:

- Made the Hats simulator faster, more extensible and more robust,
- Published Hats data in EAGLE Database (EDB) format (both the old format and the new), in Comma Separated Value (CSV) format and in "Lispy" format,
- Continued to develop the Bayesian Blackboard,
- Ran comparison experiments of algorithms from NYU, 21st Century, ISI, KDL and CMU. These verified that a Blackboard Architecture could indeed combine typical intelligence algorithms[1] and perform better that the algorithms did on their own,
- Maintained the Hats community via an online web site, a Wiki and mailing lists,
- Developed algorithms for population generation and suspicion scoring,
- Analyzed a guilt by association model.

The Hats simulator and Information Broker have proven to be excellent tools for experimenting with algorithms and analysis techniques.

The Hats Simulator and the Information Broker

Hats is a virtual world in which agents (called hats) move around, go to meetings, acquire capabilities, do business, and, for a small subpopulation of agents, do harm. Agents move on a two-dimensional board that has only two kinds of locations: Beacons are high-value places that terrorist agents would like to destroy, other locations have low value. All beacons have a set of attributes, or vulnerabilities, corresponding to the capabilities agents carry. To destroy a beacon, a terrorist task force must be in possession of a set of capabilities that match a beacon's vulnerabilities, as a key matches a lock. In general, these sets of capabilities are not unique to terrorists, so one cannot identify a terrorist task force from its constituent capabilities alone. Hats serves as an unclassified proxy for real Intelligence analysis. For more information, see [2].

The Information Broker provides a buffer between the data produced by Hats and intelligence analysts. Like the real world, information can be retrieved from this buffer only by paying for it and the quality of the information improves as more is paid. The Information Broker provides Hats with a model of utility: how much should we pay for information in order to prevent an attack?

In this effort, we have extended Hats and the Information Broker by dramatically increasing simulation speed to the point were we can generate populations of hundreds of thousands of agents in minutes and run a world of 100,000-agents for 2000-ticks in less

[1] Typical algorithms are group finders like K-Groups or KOJAK and suspicion scorers like NYU's relational classifier or ISI GBA.

than 5 hours[2]. If a tick represents a day then we can quickly simulate years of terrorist activity. In addition, we have improved the architecture of Hats and the Information Broker so that it is easier to include new adversarial planners, population generators and world models. Finally, we have worked to make the Hats code base more portable to other Lisp dialects such as OpenMCL, LispWorks, Franz Lisp, and SBCL.

The Hats mini-TIE

Community Mechanics

We have provided a Hats web site (at eksl.cs.umass.edu/hats/) and a collaborative web site (at eksl.cs.umass.edu/hats/cliki/)[3]. These have been used to introduce the Hats simulator and to facilitate discussion on the data sets we have produced.

Data Export

Significant effort was required to develop data export routines to support the original EDB schema because of its convoluted nature. We translated the Hats ontology into that of the EDB schema and developed Lisp/SQL routines to perform the conversion. Because this work proceeded more slowly than planned, we also developed simple CSV and Lispy export routines so that the Hats mini-TIE community could see our data quickly.

Shortly after we had completed the original EDB export routines, EAGLE instituted a major change in the EDB schema. It was difficult to fit the Hats ontology into the new schema but we persevered and developed new data export routines. In the process, we added the ability to produce data with customizable noise (this included additional records, changed records and missing records). Several data sets were produced and used in the

Bayesian Blackboard

EKSL has continued to develop the Bayesian Blackboard. This technology aims to combine rational Bayesian statistical techniques with dynamic Blackboard based execution control in order to achieve fluid mixed initiative analysis of incoming intelligence data. During the EAGLE project, EKSL first prototyped a proof of concept system and used it to analyze data from the Capture the Flag war-gaming system and the Hats simulator. Lessons from the initial version were then fed back into a second system which is still under development [15].

Algorithms and Experiments

In order to validate our Bayesian Blackboard design, we used Hat simulator data to run two experiments. Both provided additional proofs of the Bayesian Blackboard concept:

---

[2] Speed results are based on using Macintosh Common Lisp 5.0 running under OS X 10.3 on a 1.25-Mhz G4 Powerbook.

[3] The collaborative web site is set up as a “wiki”, the generic term for an open, generally free form web authoring system. See http://www.c2.com/cgi/wiki?WelcomeVisitors for more details.

the first examined scalability within the Blackboard; the second let us work closely with our partners to produce a "manual" Blackboard.

SCALING THE BLACKBOARD

Like all Blackboards, the Bayesian Blackboard contains data about the domain – in this case Hats – and control information [14, 11]. In particular, the Blackboard database contains primary and secondary data about the entities in the Hats simulator and Bayesian Fragments representing facts and inferences about those entities. Control is achieved via fragment combination and graph matching, Bayesian statistical inference, the creation and obviation of hypothesis, call outs to external algorithms (to provide additional, perhaps non-Bayesian analysis) and human input. Our Bayesian fragments closely follow the work of Kathleen Lansky [9, 10] in that they closely represent first order logical expressions. For example, we start with statements like:

- Taskforce TF is about to attack beacon B
- Taskforce TF is malicious
- Taskforce TF is close to beacon B (within striking distance)

And convert them into Bayesian fragments. These fragments are then combined to produce networks such as that in Figure 1.

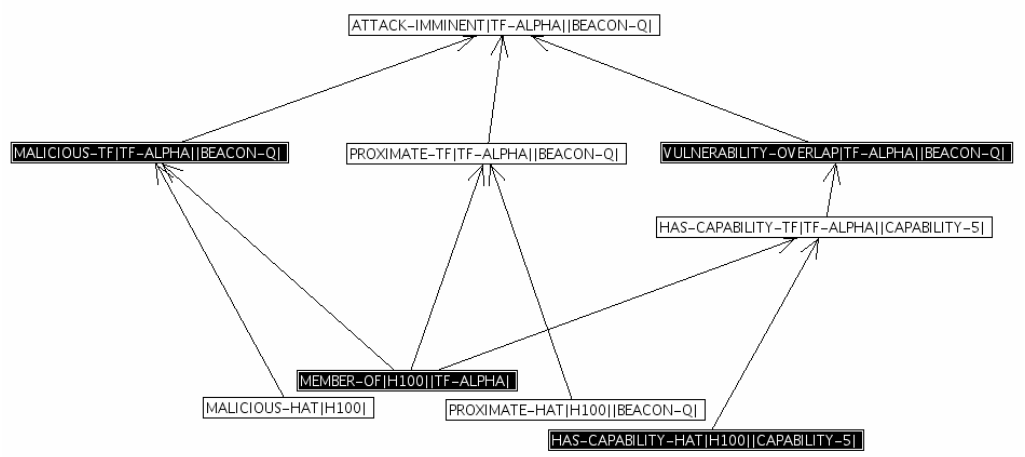


Figure 1: Bayesian network representing the probability of imminent-attack of beacon-Q by taskforce Alpha, based on information from hat-1

All logical formalisms suffer from the combinatorial curse when applied to real world problems and Blackboard systems are no exception. For example, looking for task forces is a dreadful proposition in even a very small hats world with 1000 benign hats and 100 terrorists (half known, half covert). Suppose we knew that all task forces consist of exactly four members, we can ask how many nodes we would need to post on the Blackboard in order to find threatening groups. Table 1 provides a summary of the answer starting with the naïve estimate of 1100 hats choose 4. It then goes on to show how information from the Blackboard knowledge sources can greatly reduce the total number of task forces that must be examined. Clearly, Blackboard control is essential to making even simple problem tractable.

Scenario	Taskforce Count	Reduction
1100 hats choose 4	60,671,970,975	N/A
65 high suspicion hats choose 4 (via NYU)	677,040	89,000
66 terrorist organization hats (via KOJAK group finder)	720,720	84,000
Each of 3 groups has 36 members	176,715	3.9
Suspicious taskforces (average score ≥ 0.7)	152,901	1.1
Suspicious taskforces with no more than two known terrorists	96,768	1.6
Suspicion taskforces with no known terrorists	1591	60.8

Table 1: Taskforce Count for small Hat world under various conditions

WORKING TOGETHER ON THE MANUAL BLACKBOARD

In the manual Blackboard experiment, CMU and ISI/USC provided group finding algorithms; NYU and the KDL group at the University of Massachusetts suspicion scorers and the University of Texas at Austin and 21st Century provided pattern analysis. In the experiment, we produced Hats data from the simulator and passed it along to each group for analysis. We then shared the analysis results amongst the groups so that they could use this additional information for further analysis. Our hypothesis was that we would find improved results when groups were allowed to share information. If this simple manual Blackboard failed to show that algorithms can improve one another's results, there would be little point in building more elaborate systems.

We provided snapshots of data from the Hats simulator taken just before a beacon was attacked (at ticks 673, 1263 and 1767). Our goal was to see how well the various algorithms performed with additional data and whether or not the algorithms could help one another. Table 2 depicts the data characteristics:

TICKS	672	1262	1766
Nodes	1100	1100	1100
Links	3918	7248	10,071
Average degree	7.1	13.2	18.3
Clustering Coefficient	0.44	0.31	0.27

Table 2: Dataset properties.

Suspicion Scoring

The KDL Group at the University of Massachusetts, Amherst and Sofus Macskassy at NYU examined the data with their suspicion scoring algorithms. Their analyzed the data

both with and without group information from CMU GDA group detector. The results are shown in tables 3 and 4.

	Benign Recall	Terrorist Recall	Overall Accuracy	AUC
0-672				
	1.00	0.91	0.99	0.98
with Groups	1.00	0.91	0.99	0.98
0-1262				
	1.00	0.91	0.99	0.96
with Groups	1.00	0.96	0.99	0.99
0-1766				
	0.99	0.91	0.98	0.99
with Groups	0.98	0.93	0.98	0.98

Table 3: KDL F-scores with and without group data

		AUC
0-672		
		0.92
	with Groups	0.86
0-1262		
		0.95
	with Groups	0.80
0-1766		
		0.96
	with Groups	0.80

Table 4: NYU F-scores with and without group data

KDL found that the group information improved accuracy somewhat without disturbing recall. On the other hand, NYU saw a drop in overall AUC. Both algorithms performed exceptionally well on the sample datasets which indicates that we may be seeing a ceiling effect. More study is needed using more difficult datasets in order to tease out what we are actually seeing.

Group Finding

We using ISI KOJAK Group finder both with and without suspicion score information from NYU and KDL. We used the suspicion scores in several different ways:

- As a pre-processing filter: drop hats from the list if the suspicion scorer classifies it as a “good guy” with high probability.
- As a post-processing filter: remove good guys before looking for groups
- As a seeding technique: use highly suspicious hats as seed members for groups

All of these methods show a 2-3% improvement over and above KOJAK's already high performance. Once again, further analysis is needed to understand what we are seeing in part because of ceiling effects.

Another possible problem is that Hats data is intentionally simple: there just is not that much for these algorithms to use. It may be that each technique is essentially "using up" all of the available information in the data. We are still in the process of attempting to quantify this idea so that it can be analyzed.

Pattern Finding

Lastly, 21st Century analyzed Hats data with their pattern finding algorithms. They only looked at the smallest data set (from ticks 0 through 672) and searched for a pattern of groups whose member-capabilities cover all of the capabilities associated with a particular beacon (recall that these are the groups that can potentially destroy the beacon). Out of 33 beacons, they found 7 pattern beacons with matching patterns. There were 19 final meetings at these 7 beacons (as compared to 84 final meetings for all 33 beacons). Table 5 shows the results of comparing the threat groups found by 21st Century to all of the task forces assigned to beacons.

Precision	Recall	Hit count	False positives	Miss count
0.87	1.0	10.86	1.57	0.0

Table 5: 21st Century pattern finding results

The advantage here is that we have actually used both capability and link information to find task forces which may allow for a much stronger focus. On the negative side, the patterns found both terrorist and benign task forces, missed 26 out of 33 beacons and did not make use of temporal data such as capability trading.

We still need to explore combining suspicion scoring and group finding with the pattern finding technology.

Algorithms

In the process of incorporating and analyzing outside algorithms, we also developed our own suspicion scoring and group finding techniques.

SUSPICION SCORING: THE PERJURY MODEL

From the beginning, Hats has included a simple perjury model that taints individuals based on how often they meet other tainted individuals [7]. In particular, we set the perjury p of a Hat h at time t to:

$$p_h(t) = p_h(t\text{-}1) - \beta\, p_h(t\text{-}1) + \alpha(1 - p_h(t\text{-}1))z$$

Where α and β are parameters that set how quickly perjury rises and falls. The function z is based on the scores of hats in contact with h at time $t\text{-}1$. This formulation keeps the perjury score of each hat within the interval [0, 1]. We analyzed this guilt-by-association model and found both empirically and analytically that it is very sensitive to changes in

its α and β parameters: that tiny changes in them would produce radically different policy implementations – clearly an undesirable outcome. This is a simple guilt-by-association model and our results do not imply that such models have this problem. What we do show, however, is that analyzing the perturbation sensitivity of models is vitally important.

GROUP FINDING:

We also developed a novel method of finding groups of related individuals in a larger population based on the differing dynamics of classification in members and non-members. Despite its simplicity, this algorithm performed very well on data from the Hats simulator [6]. The essential insight of the algorithm is that the connectivity of group members is different from that of non-group members. If we view suspicion as a disease, then this differing connectivity translates into different epidemic dynamics for members and non-members. Surprisingly, this difference is enough to do as well as many other group finding algorithms.

Summary

In conclusion, the Hats simulator and its Information Broker have proven to be excellent tools for experimenting with algorithms and analysis techniques. The data produced can be placed on the Bayesian Blackboard and analyzed with multiple tools from multiple research groups. Hats data will provide researchers with fodder for many questions for much time to come.

References

[1] BARKER, K., BLYTHE, J., BORCHARDT, G., CHAUDHRI, V. K., CLARK, P. E., COHEN, P., FITZGERALD, J., FORBUS, K., GIL, Y., KATZ, B., KIM, J., KING, G., MISHRA, S., MURRAY, K., OTSTOTT, C., PORTER, B., SCHRAG, R. C., URIBE, T., USHER, J., AND YEH, P. Z. A knowledge acquisition tool for course of action analysis.

[2] COHEN, P., AND MORRISON, C. The hats simulator. In *Proceedings of the Winter Simulation Conference* (2004).

[3] COHEN, P. R., HEERINGA, B., AND ADAMS, N. An unsupervised algorithm for segmenting categorical time series into episodes. In *Working Notes on Pattern Detection and Discovery in Data Mining* (2002).

[4] COHEN, P. R., HEERINGA, B., AND ADAMS, N. An unsupervised algorithm for segmenting categorical time series into episodes. In *Proceedings of the IEEE International Conference on Data Mining* (2002).

[5] COHEN, P. R., AND MORRISON, C. T. The hats simulator. In *Proceedings of the 2004 Winter Simulation Conference* (2004), R. G. Ingalls, M. D. Rosetti, J. S. Smith, and B. A. Peters, Eds.

[6] GALSTYAN, A., AND COHEN, P. Identifying covert sub-networks through iterative node classi_cation. In *Proceedings of the First International Conference on Intelligence Analysis* (2005).

[7] GALSTYAN, A., AND COHEN, P. Is guilt by association a bad thing? In *Proceedings of the First International Conference on Intelligence Analysis* (2005).

[8] HANNON, A. C., KING, G., MORRISON, C., GALSTYAN, A., AND COHEN, P. Population generation for large-scale simulation. In *Proceedings of AeroSense 2005* (2005).

[9] LASKEY, K. B. Knowledge representation and model construction, 2000.

[10] LASKEY, K. B., AND MAHONEY, S. M. Network fragments: Representing knowledge for constructing probabilistic models. Morgan Kaufmann Publishers.

[11] LESSER, V., WHITEHAIR, R. C., CORKILL, D. D., AND HERNANDEZ., J. A. Goal relationships and their use in a blackboard architecture. In *Blackboard Architectures and Applications* (1989), V. Jagannathan, R. Dodhiawala, and L. S. Baum, Eds., Academic Press, pp. 9–26.

[12] MORRISON, C. T., AND COHEN, P. R. The value of noisy information. In *Proceedings of The Sixth International Symposium on Intelligent Data Analysis (IDA-2005)* (2005).

[13] MORRISON, C. T., COHEN, P. R., KING, G. W., MOODY, J., AND HANNON, A. Simulating terrorist threat in the hats simulator. In *Proceedings of the First International Conference on Intelligence Analysis* (2005).

[14] NII, H. P. Blackboard systems. In *The Handbook of Artificial Intelligence: Volume IV* (1989), P. R. C. Avron Barr and E. A. Feigenbaum, Eds., Addison-Wesley, pp. 1–82.

[15] SUTTON, C., BURNS, B., MORRISON, C. T., AND COHEN, P. R. Guided incremental construction of belief networks. In *Fifth International Symposium on Intelligent Data Analysis* (2003).

Appendix A:

A Knowledge Acquisition Tool for Course of Action Analysis*

Ken Barker[1], Jim Blythe[2], Gary Borchardt[3], Vinay K. Chaudhri[4], Peter E. Clark[5], Paul Cohen[6], Julie Fitzgerald[9], Ken Forbus[7], Yolanda Gil[2], Boris Katz[3], Jihie Kim[2], Gary King[6], Sunil Mishra[4], Clayton Morrison[6], Ken Murray[4], Charley Otstott[8], Bruce Porter[1], Robert C. Schrag[9], Tomás Uribe[4], Jeff Usher[7], Peter Z. Yeh[1]

1. University of Texas at Austin 2. Information Sciences Institute at University of Southern California 3. Massachusetts Institute of Technology 4. SRI International 5. The Boeing Company 6. University of Massachusetts at Amherst 7. Northwestern University 8. Retired Lieutenant General, U.S. Army 9. Information Extraction and Transport Corporation

Abstract

We present the novel application of a general-purpose knowledge-based system, SHAKEN, to the specific task of acquiring knowledge for military Course of Action (COA) analysis. We show how SHAKEN can capture and reuse expert knowledge for COA critiquing, which can then be used to produce high-level COA assessments through declarative inference and simulation. The system has been tested and evaluated by domain experts, and we report on the results. The generality of the approach makes it applicable to task analysis and knowledge capture in other domains. The primary objective of this work is to demonstrate the application of the knowledge acquisition technology to the task of COA analysis. Developing a system deployable in an operational environment is the subject of future work.

Introduction

The goal of the SHAKEN project is to let subject matter experts (SMEs), unassisted by AI technologists, assemble models of mechanisms and processes from components. Questions about these models can be answered both by conventional inference methods, such as theorem proving and taxonomic inference, and by more task-specific methods, such as simulation and analogical reasoning. We believe that the assembly of components instantiated to a domain is a natural way for SMEs to create knowledge base content.

This paper describes the application of SHAKEN to the acquisition and use of knowledge needed for military Course of Action (COA) analysis. We begin with a technical overview of SHAKEN. We then describe the COA application, and give an overview of its solution using SHAKEN. For each aspect of the solution, we describe the technical challenges faced, and how we addressed them. We conclude with an evaluation of our approach, and directions for future work.

Functional Design of SHAKEN

The SHAKEN system has the following functional units, shown in Figure 1: a knowledge base (KB), an interface for entering knowledge, a set of tools for verifying and using knowledge, and a Web-based interaction manager. The KB, also called the *component library*, or CLIB [3], is a collection of components representing (a) general knowledge about common physical objects and events, states of existence, and core theories, including time, space, and causality, and (b) more specialized knowledge about particular domains, including micro-biology, chemistry, military units, military equipment, and terrain.

By a component, we mean a coherent set of axioms that describe some abstract phenomenon (e.g., the concept of *invade*) and are packaged into a single representational unit. Our claim is that a small number of predefined components is sufficient to let SMEs assemble models of virtually any mechanism or process. These components are mostly domain independent, but their assembly and specialization can create domain-specific representations.

The main task of the knowledge entry interface is to let SMEs assemble the right KB components, by connecting predefined elements of the component library. This is performed through a graphical interface, where SMEs assemble components by manipulating graphs. Axioms are automatically derived from the graphical representation, so the SMEs do not have to be trained in formal logic [8].

SHAKEN supports several different methods for using knowledge. Declarative inference, performed using the Knowledge Machine knowledge representation system (KM) [7], is the most common approach for using knowledge. Normative simulation is used to exercise the process knowledge in the system [17]. It executes each step in the process and analyzes interdependencies. Empirical simulation exercises knowledge by running a detailed simulation of a process using the Capture the Flag simulation engine [1]. An analogical reasoner, based on the Structure-Mapping engine [9], computes similarities and differences given two concept representations [21]. These methods can be invoked by a variety of means included in the question-asking interface [6]. The answers to questions are returned in an easily understood format, and the user can control the level of detail in an answer.

The interaction manager is aimed at making the knowledge entry experience seem natural. It handles limited forms of natural language input, and keeps track of

* For more information about the SHAKEN system, contact Vinay Chaudhri at chaudhri@ai.sri.com

the history of a knowledge acquisition session. A knowledge analysis module and an analogy module support the interaction manager and let SHAKEN take the initiative in helping an SME enter knowledge [17]. For example, the knowledge analysis module helps users verify and validate their process descriptions by analyzing the results from normative simulation. The vision for the interaction manager is to make the knowledge entry similar to a student/teacher interaction, where both the user and the system take the initiative at different times [19].

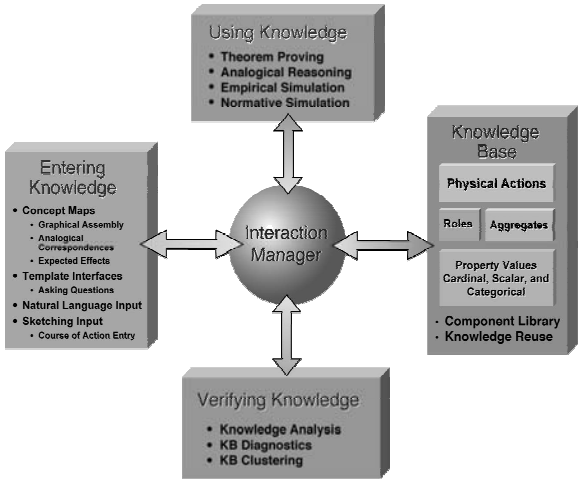


Figure 1: SHAKEN functional architecture

The KB server provides facilities for efficient storage and access of knowledge, based on KM [7]. It stores both domain-independent and domain-specific knowledge.

Knowledge verification based on normative simulation is used during knowledge entry by SMEs. KB clustering and diagnostics are used off-line both to support the development of domain-independent knowledge, and to do a post-hoc analysis of the knowledge entered by the SME.

Task: Course of Action Analysis

A military COA is a plan outline used by a commander to communicate to his subordinates one way to accomplish a mission. Normally, commanders consider several different ways to accomplish a mission, that is, several different COAs. They evaluate competing COAs using appropriate comparison criteria and decide on one to build into a complete action plan for the mission. In this paper, we consider COAs for ground military forces conducting offensive (attack) operations. The detail captured in the COA depends on the echelon. We consider here COAs at the level of a military division, a brigade, or a batallion. We consider only the COAs of friendly forces. Possible COAs for the enemy forces are not considered.

A COA specification is formulated in response to a specific situation between opposing forces and a mission directive. For purposes of description, we organize a COA specification into two parts: problem statement and solution statement. A COA problem statement consists of the following: (1) a situation sketch (on a map), indicating terrain features such as roads, rivers, lakes, hills, forests, and current Blue and Red unit placement; (2) a scenario narrative, including any details not easily captured on the map (e.g., relevant recent history, current dynamics, expected future evolution, unit status descriptions); (3) a mission specification, indicating specific forces under command, required objectives, and constraints (e.g., "Capture Objective JAYHAWK by 1400 hours tomorrow with the following restrictions in place…"); and (4) the commander's estimate of the situation.

Faced with such a problem statement, a commander must formulate a plan for his forces to accomplish the mission. He considers one or more options, or COAs. A COA solution consists of: (1) a COA sketch—an overlay on the problem statement's situation sketch, and (2) a COA narrative—a structured description stating the mission, commander's intent, desired end state, and the concept of operations, including main attack, supporting attack, fire support, and reserve. Each task in the COA must indicate what units perform what actions for what purposes.

Given enough time to consider alternatives, the commander's staff evaluates the candidate COAs in a subjective critiquing process, usually resulting in a matrix comparing the viable ones, and presents the results to the commander for a decision on the preferred COA. Commonly used COA-critiquing criteria include mission accomplishment, reserve availability, speed, simplicity, terrain use, risk, and position for follow-up operations. With help from domain experts, we created an extensive taxonomy of critiquing criteria. The COA critiquing task is to evaluate a formally represented COA with respect to key critiquing criteria. The purpose of critiquing and comparing different COAs is to help the commander decide how best to accomplish the assigned mission.

Given this definition of the COA analysis problem, the tasks to be performed were twofold: (1) given textual and graphical COA problem statements, formally represent selected elements of these in a knowledge base, and (2) author (conceive of and formally represent) knowledge to support effective COA critiques, which can then be applied to any formally represented COA solution statement.

We now briefly consider the possible deployment of a COA critiquing system. The critiquing knowledge will be entered in an Army laboratory long before the system is actually used in the field. The COA problem and solution statements will be entered at the time of actual usage of the system. Thus, when the critiquing task is performed in response to an actual need, the relevant critiquing knowledge will already be available. Given that we were developing an initial prototype, the task of entering COA problem and solution statements, and the task of authoring critiquing knowledge, are interleaved much more than they might in a situation when a COA critiquing system has been built and deployed.

Solution: Using SHAKEN to Acquire and Apply COA Critiquing Knowledge

As stated in the previous section, the overall task has two main aspects: COA authoring, and COA critiquing. With reference to the functional architecture of Figure 1, the tasks of authoring the COA and the critiquing knowledge are supported by the knowledge entry subsystem. COA authoring relies on battlespace knowledge that is built into the knowledge base. The SME enters the critiquing knowledge during development, which is stored in the knowledge base. The module focused on using knowledge supports the critiquing task. The interaction manager and the knowledge verification module play a supporting role in the overall solution of the problem.

COA Authoring

To formally author a COA, we needed to solve two problems: (1) provide a vocabulary of terms that can be used in COA authoring, and (2) provide a natural user interface for commanders.

Vocabulary for COA authoring: To support COA authoring, we need to represent military units, terrain, and military tasks. For military tasks, we developed two different representations: one suitable for declarative inference, and the other suitable for empirical simulation. Let us consider these two in more detail.

To develop representations for knowledge analysis, we leveraged the domain-independent representations in the component library to provide military-specific terms. For example, consider the military task *Canalize*. This is a tactical mission task where a military unit restricts enemy movement to a narrow zone. We represented this domain-specific action by specializing the domain-independent action *Confine*. The *Canalize* task differs from *Confine* in that its *agent* and *object* are military units, and its *base* is a piece of narrow terrain. It is similar to *Confine* in that its base plays the role of a container, and the object is inside the base after the action has been performed.

Empirical simulation requires a model of the domain and a model of the processes that occur in that domain. Our domain model is built on the University of Massachusetts Abstract Force Simulator (AFS) [2]. Military engagements are represented using circular agents moving on a coarse representation of real terrain. The agents have many properties, but most of the ones significant to military modeling (training, weapons type, troop strength, experience, and so on) are agglomerated into a single property: mass. The process model represents actions as lists of desired effects on key properties. Figure 2 shows the action model for *Defeat*, which is broken into two phases: one for the friendly forces to reach the enemy and one for the engagement. Each phase has corresponding goals for the action. The action models for the military tasks in the field manual are represented within AFS using Tapir, a general purpose, semi-declarative hierarchical agent control language that can express goals, sensors and actions using a unified syntax [18]. During each simulation run, the action models control the military agents; dynamically reacting to the changing properties of the simulation in order to achieve their goals.

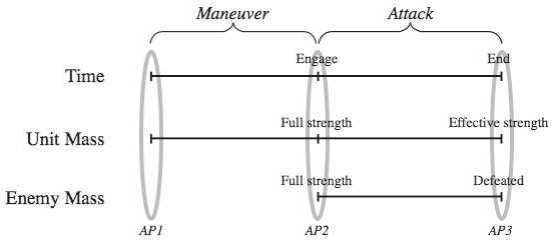


Figure 2: Action model for *Defeat*

User interface for COA authoring: We needed an interface that was as familiar to commanders as possible. Commanders work with maps and overlays to show the geography, unit locations, and military tasks. The map is usually accompanied by a textual description. The nuSketch system is explicitly designed to support COA authoring, and met this requirement very well [12], [13].

NuSketch provides a graphical interface where COA terrain, units, avenues of approach, and tasks can be described. The user can also specify the commander's intent for the overall COA and individual tasks. An example COA sketch is shown in Figure 3.

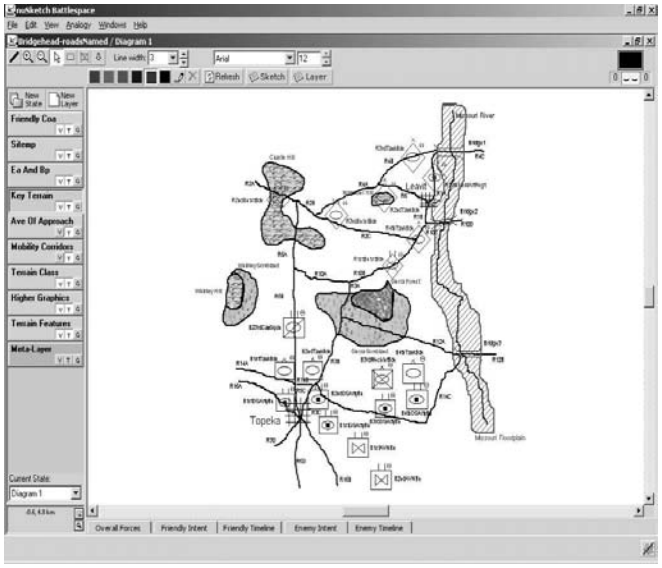


Figure 3: nuSketch COA authoring interface

NuSketch elements have a precise declarative semantics that is reflected in the SHAKEN component library ontology. Once the COA is specified in nuSketch, it is translated to a SHAKEN concept map (CMAP). The translator maps terms in the nuSketch ontology to the

corresponding terms in the SHAKEN component library. In some cases, the knowledge is processed to resolve ontological mismatches; for instance, the task timing information in nuSketch is based on the quantitative start and end times, whereas SHAKEN relies on qualitative ordering information among tasks; therefore, the translator processes the quantitative information to derive the necessary qualitative ordering.

As expected, the experts want the interface to be as easy and quick to use as their regular pen-and-paper way of doing things. The primary obstacle to achieving this was to find a suitable combination of sketching gestures, and a layout of windows that would enable rapid authoring of the COA. Currently, it takes 1 to 2 hours to author a COA. The SMEs would like to be able to do it within 15 minutes.

Critiquing Knowledge

Critiquing relies on both domain-independent and specialized knowledge. Domain-independent knowledge is leveraged as domain-specific terms are created, by specializing domain-independent terms. We will primarily discuss here the domain-specific critiquing knowledge.

Two kinds of domain-dependent critiquing knowledge were needed: (1) necessary and sufficient slot values of concepts, and (2) critiquing rules. We now consider in more detail how each was entered.

Necessary properties of concepts: The SHAKEN graphical interface is the primary means used to create the domain-specific concepts from domain-independent ones. For example, for each kind of terrain, we encoded its trafficability for each kind of unit. For each unit, we encoded the equipment it possesses, and its combat power. For each military task, we encoded how much relative combat power is generally thought to be sufficient to effectively perform this task. The tasks are encoded using a STRIPS-like language used by many AI planners [4].

As a concrete example, Figure 4 shows the representation of the concept of *Rolling-Hills*. This concept map indicates that rolling hills offer relatively unrestricted movement for armor and infantry units. See [8] for a description of how logical axioms are synthesized from graphs such as this.

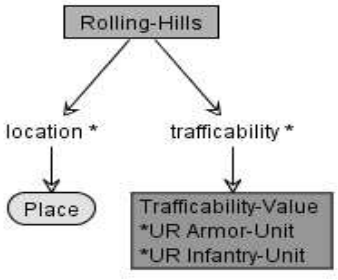


Figure 4: Trafficability definition for *Rolling Hills*

Sufficient properties of concepts: For many concepts, it is possible to define both necessary and sufficient properties. For example, if *Blue-Military-Unit* represents the class of all friendly units, then any military unit whose allegiance is *Blue* is a member of this class. A domain expert specifies the sufficient properties of a concept by annotating the graph representing the necessary properties.

The most common application of sufficient properties was to create subclasses of actions representing a specific situation, indicating a special case. For example, the required relative combat power ratio for the most general case of each military action is built into the system. However, the actual relative combat power ratio depends on the specifics of the situation. For instance, a ratio of 3 is normally desired for a general attack, but when an aviation unit attacks an armor unit, a combat power ratio of 0.5 is adequate. When a commander authors a COA, he may use the general attack action vocabulary. But, if the knowledge base includes a subclass of the attack action whose sufficient properties are that the agent is an aviation unit, and the object is an armor unit, its lower relative combat power ratio will be used whenever such a situation arises. Figure 5 shows the concept map for such a class. See [16] for more details on entering special cases of actions.

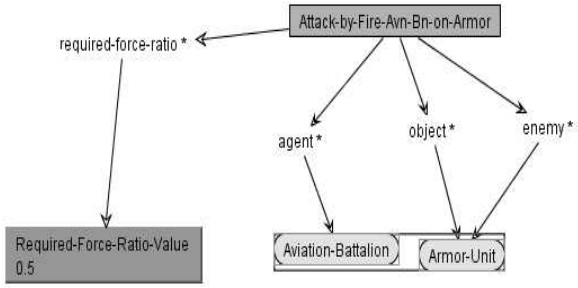


Figure 5: A special case of the *Attack* action. The nodes grouped in a box indicate sufficient properties.

Critiquing rules: We devised a special kind of rule, called a *pattern*, where the antecedent represents a collection of assertions pertaining to the situation being critiqued, and the consequent is a critiquing score on some critiquing dimension. Figure 6 shows an example pattern that rates a COA as good if some forces are kept in the reserve. The portion of the graph linked to the root with the *has-pattern* relation indicates an antecedent, and the portion linked using *critique-score* indicates the consequent of the rule.

Critique scores can be positive or negative, and a single pattern can apply to more than one critiquing dimension. Critiquing dimensions for COA patterns include such concepts as Risk, Casualties, Maneuver Effectiveness, Command and Control, Terrain Use, Preparedness for Enemy Response, Simplicity, Resource Use, and Synchronization. Applying these rules, organized by the critiquing dimensions, gives a direct rating of a COA.

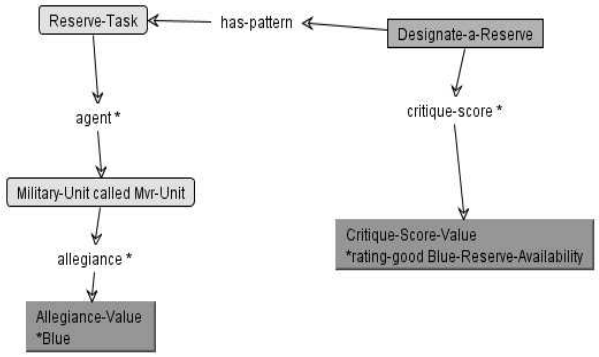


Figure 6: A pattern indicating that allocating a reserve is good for *Blue-Reserve-Availability*

Exercising Critiquing Knowledge

SHAKEN currently supports three different kinds of critiquing: declarative inference, normative simulation, and empirical simulation. (SHAKEN's analogical reasoning capabilities can also be used for critiquing [10], but this is not covered in the present paper.)

Critiquing by declarative inference: COA critiquing by declarative inference systematically finds and applies all applicable COA patterns and assigns them a score. The key technical challenge in matching patterns against a COA is that matches may not be syntactically exact. Therefore, we built a utility that can compute matches modulo a set of transformations. For example, we may know from the COA that a Blue force is in a city; we may also have a pattern saying that if an armor unit is in a city, it is poor for security of that unit (unless it is accompanied by infantry that can protect tanks in narrow streets and alleys from short-range antitank weapons). The pattern matcher will match the COA and the pattern, noticing that the Blue force has an armor unit that is in the same location. The pattern matcher contains a few hundred such transformations.

☐ Match #2 top

Score: rating-verygood **on the dimension of** Deception-Operation-Use

Score: rating-good **on the dimension of** Synchronization

Score: rating-good **on the dimension of** COA-Effectiveness

Node correspondences

Pattern	COA
the Main-Attack-Task and Engagement-Military-Task called Engage-Enemy-2	the Main-Attack-Task and Seize-Terrain-Feature called Object-640
Conducting-MA	B2ndTankBde
Supporting-MA	B4thTankBde
the Supporting-Attack-Task and Engagement-Military-Task called Engage-Enemy-1	the Supporting-Attack-Task and Seize-Terrain-Feature called Object-764

Figure 7: A report from critiquing by patterns

Figure 7 shows an example report generated by matching patterns, as presented by the SHAKEN interface. The top of the report indicates the critiquing scores. The COA being evaluated has a score of *Very Good* on the dimension of deception. The table that follows indicates which nodes in the pattern matched which nodes in the COA. For example, *B2ndTankBde* conducts the main attack, and *B4thTankBde* conducts the supporting attack.

Critiquing by normative simulation: Normative simulation critiques a COA by executing each step. It relies on the KM situation mechanism, and executes each step based on its effects (add/delete lists). It analyzes dependencies between conditions and effects, checking that the required conditions for each step are met when the step is supposed to take place, and that the expected effects of the overall process are, in fact, obtained. It also checks how different steps are related to each other, including their temporal ordering and causal relationships. The simulation reports possible errors and presents them as critiques. For instance, for each step in the COA, normative simulation computes the net relative combat power available, and compares it against the required relative combat power ratios already encoded in the system.

Figure 8 shows an example normative simulation report. In this case, one of the preconditions of a military action has failed: the given combat power ratio is not high enough to perform the given task. The net relative combat power of a military unit is computed based on the combat power of its subunits. The explanation section of the report shows in detail how the combat power was computed by combining various pieces of information, including unit equipment, default combat power, and remaining unit strength, through multiple COA steps. The user can check this explanation to see why the condition failed.

agree with the results for this step ? Yes No

Step information

Checking soft conditions

Warning: System found these conditions to be false:

Force ratio explanation

- Units involved are (B1stAVNBn)
- For unit B1stAVNBn [(Aviation-Battalion)], (allegiance is Blue), its equipment is (the AH64) so the default combat power is 2.81
- Since its remaining strength is 0.66 the relative combat power is (2.81 * 0.66) = 1.87
- Therefore, total relative-combat-power is : 1.87
- Enemies : (R2ndTankBde)
- For unit R2ndTankBde [(Armored-Brigade)], (allegiance is Red), its equipment is unknown so the default combat power is 2.56
- Since its remaining strength is 0.85 the relative combat power is (2.56 * 0.85) = 2.18
- So the available-force-ratio is : 1.87 / 2.18 = 0.86

1. **The available-force-ratio (0.86) >= The required-force-ratio (1.0)**

Click here for suggestions

Figure 8: COA critiquing by normative simulation

The combat power numbers are dynamic, and take into account how the various units undergo attrition over a period of time. The action is flagged if the actual relative combat power during an action is less than the required relative combat power. Even when the combat power exceeds what is required, the commander can use the report information to check that all the decisive points have overwhelming relative combat power ratios.

In this instance, an SME added a special case of the Attack-by-Fire action to account for this kind of situation (i.e., when an aviation battalion attacks an armored unit, a combat power ratio of 0.5 is enough). Once this special case was added, the precondition was satisfied.

Critiquing by empirical simulation: Empirical and normative simulation complement each other in SHAKEN. Simulation is used to capture complex dynamics in the COA, and to explicitly model uncertainty. For empirical simulation, SHAKEN uses the Capture the Flag (CtF) tool [1], based on the AFS abstract physics-based model of division-level engagements described earlier (see Figure 2). Once a nuSketch COA is translated to CtF, Monte Carlo simulation is performed, running the COA multiple times until statistically significant results are obtained. The data from these trials is summarized in HTML reports, showing combat power ratios and graphical snapshots of critical events (e.g., engagements) during the simulated runs.

Figure 9 shows the combat power ratio graph produced for a particular engagement during a single simulation run. The ratio increases as the Blue side gains dominance over time, indicating a Blue army victory. A chief strength of empirical simulation is unexpectedly simple: SMEs can watch their COAs unfold visually, and can immediately see flaws and strengths. The results are analyzed to construct a qualitative representation of the space of outcomes, explicitly identifying critical points.

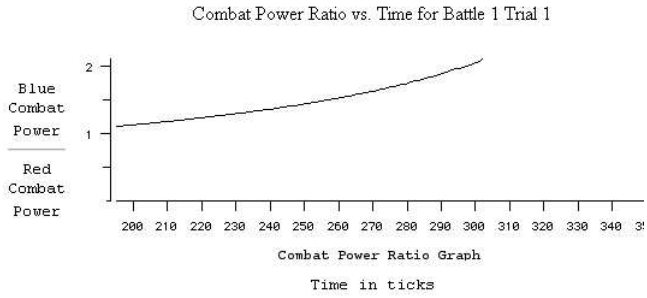


Figure 9: Output from empirical CtF simulation

Evaluation

We evaluated the system with the help of two domain experts, both of whom were retired Army officers. One had served at the rank of lieutenant general, and the other as an intelligence officer. The objective of the evaluation was twofold: to assess how effectively the knowledge acquisition capabilities of SHAKEN would work for domain experts with no training in formal knowledge representation, and to test the performance of the resulting knowledge base on the COA critiquing task.

The evaluation was conducted over 15 days. During the first 7 days, we provided hands-on training to the two subject matter experts, using an example critiquing task. The SMEs were then given a new task, in the form of a COA problem statement and its solution, expressed in textual form, and were asked to address it using the system. The SMEs were asked to encode the textual description in SHAKEN. They then authored critiquing knowledge, independent of the COAs, and used it to critique them.

Before encoding a COA, the SMEs produced a manual critique for it, to serve as a guideline for evaluating the ultimate critique to be produced by the system. Authoring the critiquing knowledge was an iterative task: the knowledge was successively refined based on the system critique, and how it differed from the manual critique.

Over the 15-day period, the SMEs authored three different COAs and 60 pieces of critiquing knowledge. The critiquing knowledge included patterns and special cases of actions. Below, we present the textual description of a few patterns authored by the SMEs during evaluation. The critiquing dimensions are shown in bold font:

*If a COA secures a piece of terrain narrower than 50 meters, it makes good **use of terrain**.*

*If the supporting attack occurs before the main attack, it is good for COA **effectiveness, mission accomplishment**, and **synchronization**.*

*If an armored unit attacks a mechanized infantry unit outside a city, it is good for **enemy maneuver engagement.***

The antecedent encodes the condition under which the pattern applies, and often includes spatial information such as terrain or unit location. In some cases, the antecedent can include negation, for example, the location of a unit not being in a city. Let us now consider two examples of special cases of actions, where the bold text represents the sufficient property of the special case:

*When **an aviation unit attacks an artillery unit,** it is sufficient to have a combat power ratio of 0.3.*

*While **seizing a bridge**, it is sufficient to have a combat power ratio of 0.3.*

These example patterns and special cases of actions show that SMEs with very little training in knowledge representation were able to author nontrivial pieces of critiquing knowledge. In particular, the first-order logic formalization of this knowledge, synthesized automatically from the graphs by SHAKEN, includes quantified variables, implications, negation, and, in the case of special cases of actions, concept definitions (bi-directional implications). These formal structures are clearly beyond anything that the SMEs could encode directly. In addition,

through the constraints imposed by the graphical interface (e.g., guiding the SME to select concepts from the existing ontology, restricting the choices of relations to only semantically valid ones), the SMEs formalized their knowledge in conformance with SHAKEN's underlying ontology. This illustrates the key achievement of this work, namely, a significant enhancement of the SME's ability to articulate formal knowledge, in a way consistent with, and building upon, the preexisting knowledge in the system.

We tested the empirical simulation on the COAs authored by the SMEs. Monte Carlo summaries of mass lost and goals achieved over multiple simulations showed clear differences between these COAs. In addition, the COAs that we felt were most dangerous had the greatest amount of variance in their outcome. This highlights one of empirical simulation's greatest strengths: the ability to go beyond static analysis and focus instead on the dynamics of multiple concurrent processes.

Despite these achievements, we encountered several limitations. The most significant problem is to translate natural but informal domain concepts (e.g., "sufficient force", "flank", "contour", "overwhelm") into a computable form (e.g., in terms of coordinates and distances), a prerequisite for machine reasoning about the domain. While SHAKEN provides good support for entering formal knowledge once that conceptual translation is made, it provides little help with the translation in the first place. This turned out to be the most notable challenge for the SMEs. It is exacerbated in the COA domain, where many important concepts are spatial in nature, but particularly difficult to pin down precisely in formal terms.

Second, although the interface helps SMEs enter knowledge in terms of the existing ontology, there is still potential for SMEs to make mistakes. For example, they sometimes used negation in a way that differed from their intent, without realizing that the semantics of what they encoded was subtly different (e.g., one SME encoded "an attack not on a city is good", intending to encode "no attack on a city is good"). More proactive checking and validation of SME inputs would help identify and correct such errors.

As additional evaluation data, at the end of the 15-day period we compared the SHAKEN critiques produced using an SME's formally encoded knowledge with the manual critiques written by the same SME. Our goal was to check that the SME's encoded knowledge was to some extent "reasonable" compared with his ideal solution (the manual critique), that is, to check that the SME's rules were not simply "formal nonsense". The SMEs were asked to assign a correctness score on a five-point scale (-2 to +2) to the results produced by SHAKEN using their encoded knowledge. A score was given to each critiquing dimension that the SME considered relevant to the particular COA.

Of the 16 relevant critiquing dimensions for one of the representative COAs, the system critique received a score of +2 for 8 of the dimensions; for 3, a score of +1; for 4, a score of –1; and for 1, a score of -2. Although many other factors influence these scores (e.g., the inherent knowledge representation and reasoning capacity of SHAKEN itself), the results indicate that the SME was able to enter at least some of his knowledge with a reasonable degree of accuracy and fidelity.

The SMEs' overall assessment was that a COA analysis capability such as the one we tested could ultimately be very useful in solving operational problems: The software can work through tedious details and double-check all potential COAs, especially when the commanders are tired, under pressure, and under time constraints.

Although our goal is to break new ground in knowledge acquisition technology, rather than to specifically critique COAs, it is nevertheless interesting to consider what it would take for the COA-critiquing application of SHAKEN, using SME-entered knowledge, to reach a sufficiently mature level for deployment. The technology requires numerous enhancements before it comes close to being deployable. For example, a library of a few hundred patterns and special cases of action will have to be built before the system starts producing non-obvious critiques that add value to what a commander can quickly determine with a visual inspection of a COA. One way to drive such a knowledge base construction is to work with a sizable collection of case studies [23] that will provide concrete test cases, a well-defined scope for knowledge entry, and clear performance criteria. The detail captured in the normative simulation can also be improved, giving special attention to simulating concurrent events.

Related Work

In previous work, we developed an extensive ontology of plan evaluation and plan critiquing [5]. In another previous study, we evaluated nuSketch as a COA authoring tool, and demonstrated that COAs authored using nuSketch were comparable in quality to ones authored with more traditional methods [22].

In the present work, the main innovations are: (a) using the plan critiquing ontology in conjunction with normative simulation; (b) acquiring critiquing knowledge in the form of patterns and necessary and sufficient conditions for actions; and (c) showing that the system can exhibit some level of COA critiquing competence, through declarative inference, normative simulation, and empirical simulation.

There has been significant work in building interactive plan authoring environments [20], but it has not addressed the specific problem of COA critiquing. The use of patterns for COA critiquing was demonstrated in [11], which let experts select subsets of a COA sketch to generate critiques that could be subsequently applied via analogy or as rules. However, that system only used information explicitly represented in the sketch, whereas a broader range of knowledge can be used in SHAKEN patterns.

Future Work

Work is under way to address many of the limitations identified in the previous section. For example, we are

making extensions to nuSketch to support richer COA descriptions. Similarly, we are implementing the normative simulation of concurrent events.

We are also developing a suite of capabilities that will let SHAKEN users enter, organize, and retrieve knowledge using English. These capabilities make use of the START [14] and Omnibase [15] systems. To perform knowledge entry, the user enters a sentence or phrase, which is parsed into a concept map representation similar to that used within SHAKEN. Through an interactive dialog between the user and the system, this concept map is refined into a SHAKEN concept map, which is added to SHAKEN' s knowledge base. Using a similar approach, English questions are translated into concept map patterns, which are then used to identify matching concepts within SHAKEN' s knowledge base.

Summary

We presented the application of a general-purpose knowledge-based system, SHAKEN, to the specific task of military Course of Action (COA) analysis. We showed how SHAKEN can capture and reuse expert knowledge for COA critiquing, and produce a high-level assessment of a COA through declarative inference and simulation. The system has been used and evaluated by domain experts. The generality of the approach makes it applicable to knowledge capture for task analysis in other domains.

Acknowledgments

We thank Commander Dennis Quinn and Lt. Gen. Len Wishart for serving as the domain experts for the evaluation, and Murray Burke for his encouragement and support for this work. This research was supported by DARPA's Rapid Knowledge Formation project under contract N66001-00-C-8018.

References

1. Atkin, M., G.W. King, D. Westbrook, B. Heeringa, A. Hannon, and P. Cohen. *SPT: Hierarchical Agent Control: A Framework for Defining Agent Behavior*. In *Fifth Intl. Conf. on Autonomous Agents*, p. 452-432, 2000.
2. Atkin, M.S., D.L. Westbrook, P.R. Cohen, and G.D. Jorstad. *AFS and HAC: Domain-General Agent Simulation and Control*. In *Workshop on Software Tools for Developing Agents, AAAI-98*, p. 89-95, 1998.
3. Barker, K., B. Porter, and P. Clark. *A Library of Generic Components for Composing Knowledge Bases*. In *International Conference of Knowledge Capture*, 2002.
4. Blythe, J. *SHAKEN Action Description Language*. Technical Report, Information Sciences Institute, University of Southern California, 2002.
5. Blythe, J. and Y. Gil. *A Problem-Solving Method for Plan Evaluation and Critiquing*. In *Intl. Knowledge Acquisition Workshop*. Banff, 1999.
6. Clark, P., K. Barker, B. Porter, A. Souther, V. Chaudhri, S. Mishra, J. Thomere, J. Blythe, J. Kim, P. Hayes, K. Forbus, and S. Nicholson. *A Modified Template-Based Approach to Question-Answering from Knowledge Bases*. Technical Report, SRI International, Menlo Park, CA, 2002.
7. Clark, P. and B. Porter. *KM -- The Knowledge Machine User Manual*. Technical Report, U. of Texas at Austin, 1999.
8. Clark, P., J. Thompson, K. Barker, B. Porter, V. Chaudhri, A. Rodriguez, J. Thomere, S. Mishra, Y. Gil, P. Hayes, and T. Reicherzer. *Knowledge Entry as Graphical Assembly of Components*. In *Intl. Conf. on Knowledge Capture*, 2001.
9. Forbus, K., R. Ferguson, and D. Gentner. *Incremental Structure Mapping*. In *Proc. Cognitive Science Society*, 1994.
10. Forbus, K., R. Ferguson, and J. Usher. *Towards a Computational Model of Sketching*. In *Intelligent User Interfaces Conference*. Santa Fe, New Mexico, 2001.
11. Forbus, K., T. Mostek, and R. Ferguson. *An Analogy Ontology for Integrating Analogical Processing and First-Principles Reasoning*. In *IAAI-02*, 2002.
12. Forbus, K. and J. Usher. *Sketching for Knowledge Capture: A Progress Report*. In *Intelligent User Interfaces*. 2002.
13. Forbus, K., J. Usher, and V. Chapman. *Sketching for Military Courses of Action Diagrams*. In *Proceedings of Intelligent User Interfaces Conference*. Miami, FL, 2003.
14. Katz, B. *Annotating the World-Wide Web using Natural Language*. In *5th RIAO Conference on Computer Assisted Information Searching on the Internet*, 1997.
15. Katz, B., S. Felshin, D. Yuret, A. Abrahim, J. Lin, G. Marton, A.J. McFarland, and B. Temelkuran. *Omnibase: Uniform Access to Heterogeneous Data for Question Answering*. In *7th International Workshop on Applications of Natural Language to Information Systems*, 2002.
16. Kim, J. and J. Blythe. *Supporting Plan Authoring and Analysis*. In *Intelligent User Interfaces*. Miami, FL, 2003.
17. Kim, J. and Y. Gil. *Knowledge Analysis on Process Models*. In *17th Intl. Joint Conf. on Artificial Intelligence (IJCAI-2001)*, p. 935-942, 2001.
18. King, G.W., M.S. Atkin, and D. Westbrook. *Tapir: The Evolution of an Agent Control Language*. In *First Conference on Autonomous Agents and Multiagent Systems*, 2002.
19. Mishra, S., A. Rodriguez, M. Eriksen, V. Chaudhri, J. Lowrance, K. Murray, and J. Thomere. *Lightweight solutions for user interfaces over the WWW*. In *International Lisp Conference*. San Francisco, CA, 2002.
20. Myers, K. *Strategic Advice for Hierarchical Planners*. In *Intl. Conf. on Knowledge Representation and Reasoning*, p. 112-123, 1996.
21. Nicholson, S. and K. Forbus. *Answering Comparison Questions in SHAKEN: A Progress Report*. In *Spring Symposium on Mining Answers from Text and Knowledge Bases*. Stanford, CA, 2002: AAAI.
22. Rasch, R., A. Kott, and K.D. Forbus. *AI on the Battlefield: An Experimental Exploration*. In *Innovative Applications of Artificial Intelligence*. Edmonton, Canada, 2002.
23. Schmitt, M.J.F., USMCR, *Mastering Tactics: A Tactical Decision Games Workbook*. Marine Corps Gazette. 1994, Quantico, VA: Marine Corps Association.

Proceedings of the 2004 Winter Simulation Conference
R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

Appendix B:

THE HATS SIMULATOR

Paul R. Cohen
Clayton T. Morrison

Center for Research on Unexpected Events (CRUE)
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292-6601, U.S.A.

ABSTRACT

The Hats Simulator is designed to be a lightweight proxy for many intelligence analysis problems, and thus a test environment for analysts' tools. It is a virtual world in which many agents engage in individual and collective activities. Most agents are benign, some intend harm. Agent activities are planned by a generative planner. Playing against the simulator, the job of the analyst is to find harmful agents before they carry out their plans. The simulator maintains information about all agents. However, information is hidden from the analyst and some is expensive. After each game, the analyst is assessed a set of scores including the cost of acquiring information about agents, the cost of falsely accusing benign agents, and the cost of failing to detect harmful agents. The simulator is implemented and currently manages the activities of up to one hundred thousand agents.

1 INTRODUCTION

The Hats Simulator was designed originally to meet the needs of academic researchers who want to contribute technology to Homeland Security efforts but lack access to domain experts and classified problems. Most academic researchers do not have security clearances and cannot work on real data, yet they want to develop tools to help analysts. In any case, real data sets are expensive: They cost a lot to develop from scratch or by "sanitizing" classified data. They also are domain-specific, yet much of the domain expertise is classified. Because data sets are expensive, many that have been made available to researchers are relatively small and the patterns to be detected within them are fixed, few, and known, so working with these data sets is a bit like solving a single "Where's Waldo" puzzle. Sometimes there also is the problem that real data sets model "signal" (terrorist activities) not "noise" (everything else) yet extracting signal from noise is a great challenge. Data sets in general are static, whereas data become available to analysts over time. It would be helpful to have a data *feed*, something that generates data as events happen. To validate analysts tools, it would be helpful to have a generator of terrorist and non-terrorist activities. The generator should be parameterized for experimental purposes (e.g., varying the distinctiveness of terrorist activities, to make them more or less easily recognizable as such); and it should come up with novel activities, requiring analysts and their tools to both recognize known patterns and reason about suspicious patterns.

Hats is home to thousands of agents (hats) which travel to meetings. Some hats are covert terrorists and a very few hats are known terrorists. All hats are governed by plans generated by a planner. Terrorist plans end in the destruction of landmarks. The object of a game against the Hats simulator is to find terrorist task forces before they carry out their plans. One pays for information about hats, and also for false arrests and destroyed landmarks. At the end of a game, one is given a score, which is the sum of these costs. The goal is to play Hats rationally, that is, to catch terrorist groups with the least combined cost of information, false arrests, and destroyed landmarks. Thus Hats serves as a testbed not only for analysts' tools but also for new theories of rational intelligence analysis. Hats encourages players to ask only for the information they need, and to not accuse hats or issue alerts without justification.

The Hats simulator is very lightweight: Agents have few attributes and engage in few elementary behaviors; however, the number of agents is enormous, and plans can involve simultaneously many agents and a great many instances of behaviors. The emphasis in Hats is not domain knowledge but managing enormous numbers of hypotheses based on scant, often inaccurate information. By simplifying agents and their elementary behaviors, we de-emphasize the domain knowledge required

to identify terrorist threats and emphasize covertness, complex group behaviors over time, and the frighteningly low signal to noise ratio.

The Hats Simulator consists of the core simulator and an information broker. The information broker is responsible for handling requests for information about the state of the simulator and thus forms the interface between the simulator and the analyst and her tools (see Figure 1). Some information has a cost, and the quality of information returned is a function of the "algorithmic dollars" spent. Analysts may also take actions: they may raise beacon alerts in an attempt to anticipate a beacon attack, and they may arrest agents believed to be planning an attack. Together, information requests and actions form the basis of scoring analyst performance in identifying terrorist threats. Scoring is assessed automatically and serves as the basis for analytic comparison between different analysts and tools. The simulator is implemented, manages the activities of up to ten thousand agents, and is a resource to a growing community of researchers.

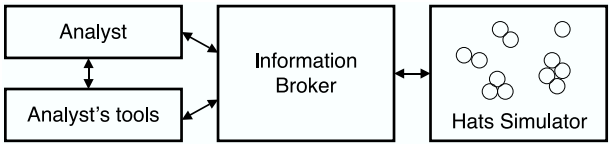


Figure 1: Information Broker Interface to Hats Simulator

The following sections outline the Hats domain, including how we generate populations of hats and how the planner schedules meetings for hats to attend. We describe the information request framework, the actions the analyst may take, and scoring. We conclude with a discussion of the future of the Hats Simulator.

2 THE HATS DOMAIN

The Hats Simulator is a virtual world in which agents move around, go to meetings, acquire capabilities, do business, and, for a small subpopulation of agents, do harm. Agents move on a two-dimensional board which has only two kinds of locations: Beacons are high-value places that terrorist agents would like to destroy, other locations have low value. All beacons have a set of attributes, or vulnerabilities, corresponding to the capabilities that agents carry. To destroy a beacon, a task force of agents must be in possession of a set of capabilities that match the beacon's vulnerabilities, as a key matches a lock. In general, these sets of capabilities are not unique to terrorists, so one cannot identify a terrorist task force from its constituent capabilities, alone.

Henceforth, agents are called hats[1] and identified as benign and terrorist; overt and covert are subcategories of terrorist hats. In general, benign hats outnumber terrorists by orders of magnitude.

Some agents are known, a priori, to intend harm – they are "known terrorists" – others are covert. This is modeled easily by assigning each agent a true and an advertised hat class:

	True Hat Class	Adv. Hat Class
Benign	Benign	Unknown
Known Terrorist	Terrorist	Terrorist
Covert Terrorist	Terrorist	Unknown

The Hats Simulator knows the true class of each hat, and it plans agents' activities accordingly, but analysts must infer hat class from how agents behave. While agents that advertise terrorist hats are "known terrorists," a very small fraction of agents that advertise an unknown class are also terrorists. They are the ones to worry about.

2.1 Organizations and Population Generation

Hats populations consist of known terrorist hats, covert terrorist hats and benign hats. All hats are members of at least one organization; some belong to many. There are two types of organizations. Terrorist organizations are made up of only known and covert terrorists. Benign organizations, however, may contain any kind of hat – that is, while known and covert terrorists must be members of at least one terrorist organization, they may also be members of benign organizations.

Hats populations may be built by hand or generated by the Hats Simulator. Because the constitution of a population affects the difficulty of identifying covert terrorists, population generation is parameterized. The `organization-overlap` parameter, a real number between 0 and 1, determines the percentage of hats in each organization that are members of other organizations. For example, if `organization-overlap` is 0.4, then 40% of the members of each organization are also members of other organizations, but the remaining 60% are only members of their native organization. The number of organizations an overlapping hat may belong to is determined by an exponential random number (thus, overlapping 3 organizations is rare, 4 is very rare, 5 is extremely rare, etc., ...). The population genera-

[1]The "hats" name is an allusion to the classic spaghetti western, in which the villain and hero are identifiable by the color of their hat.

tor manages overlap so that the `organization-overlap` percentage is as close as possible to its parametric value.

The total numbers of known terrorist, covert terrorist and benign hats in the population are determined by the `num-terrorists`, `num-coverts` and `num-benigns` parameters, respectively. Known and covert terrorists must be members of at least one terrorist organization and may also be members of benign organizations. Benign hats, on the other hand, may only be members of benign organizations. Not all organizations have the same number of members. The variable `covert-org-members-ratio` represents the ratio of covert terrorist hats assigned to each terrorist organization and `benign-org-members-ratio` represents the ratio of benign hats to each benign organization.

Assignments of hats to organizations (respecting the parameters for organization-overlap, organization members ratios, and the numbers of hat types) takes place before any actual hats are created. Once assignments have been determined, the hats themselves are generated and given their organization assignments. At this time, each hat is also assigned a native capability, which the hat will carry throughout the simulation, and a set of "traded" capabilities which are temporary, expiring after some number of ticks (e.g., within 40 ticks). Hats are also assigned random locations in the Hats world game board.

2.2 Meeting Generation

Hats act individually and collectively, but always planfully. In fact, the actions of hats are planned by a generative planner. Benign hats congregate for commerce and pleasure at locations including beacons. Terrorist hats meet, acquire capabilities, form task forces, and attack beacons. Several hats might plan to visit a beacon, and might collectively have the capabilities to destroy the beacon, yet are benign. Or, one covert terrorist might plan to visit three known terrorists in succession, acquiring from each a capability that threatens a beacon; and yet might remain dormant, approaching no beacon, for a time.

Each organization has a generative meeting planner associated with it that plans tasks for its members. A *task* is a set of meetings planned to deliver a set of capabilities to some goal location in the Hats World. Hats that participate in a task are *reserved*. Hats not part of a task are *free*. At each tick each organization has a chance of beginning a new task according to the probability specified by the `p-start-new-task` parameter. When a new task is started, the Hats meeting planner selects a subset of hats from the free hats of the organization. This subset of hats is called a *taskforce*. The size of the taskforce is determined by the `num-in-meetings` parameter. The meeting planner also selects a coordinate in the Hats World game board as the *target location* of the task. With probability specified by the `p-beacon-meeting` parameter, the planner will select a beacon location as the task target. Otherwise a random Hats World coordinate is selected. If a beacon is the task target, then the set of vulnerabilities of the beacon determines the set of capabilities the taskforce must bring to the target. If the target is not a beacon, then a random set of capabilities is selected – the size of the set of random capabilities is determined by the `num-requirements` parameter. The set of capabilities the taskforce must bring to the task target is referred to as the taskforce's *required capabilities*. The taskforce members may or may not already possess the required capabilities.

In fact, if the taskforce members generally do not have all these capabilities, then the meeting planner can construct an elaborate "shell game" in which capabilities are passed among hats at a long sequence of meetings, culminating in the fatal meeting at the target. By moving capabilities among hats, the planner can mask its intentions. It certainly is not the case that, say, half a dozen hats with required and known capabilities march purposefully up to a beacon. Instead, the hats with the required capabilities pass them on to other hats, and eventually a capable task force appears at the beacon.

Once the taskforce, target location, and required capabilities have been chosen, the meeting planner creates a set of meetings designed to ensure that the taskforce acquires all of the required capabilities before going to the target location. The meeting planner accomplishes this by constructing a *meeting tree*. Figure 2 shows an example meeting tree, where the contents of each box represent the hats participating in a meeting. The tree is "inverted" in the sense that the root is the last meeting, with branches from the root representing parent meetings that take place prior to the target meeting – Figure 2 depicts the temporal ordering of meetings by directed arrows. At this point, the meeting planner incrementally fills-out the meeting tree, starting with the final meeting. The final, root meeting takes place at the target location and involves all of the taskforce hats. The parent meetings of the final meeting each have one taskforce member. The locations of all other meetings added to the meeting tree are selected randomly.

The meeting planner selects a second set of hats (from the organization's free hats) that carry required capabilities that the taskforce does not currently carry; these hats are called *resource hats*. Each of the resource hats are randomly assigned to taskforce members. Meetings between resource hats and taskforce members are called *resource meetings*. Resource meetings are added to the meeting tree as follows. The planner traverses

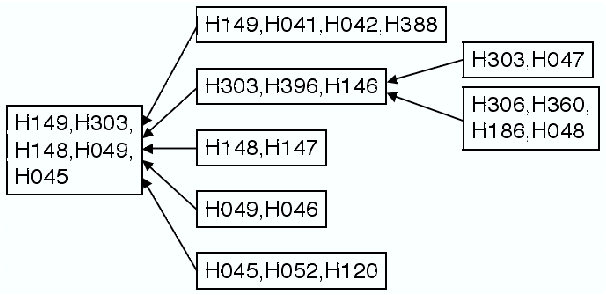


Figure 2: Example meeting tree

a branch of the meeting tree which a taskforce member originates from (initially, these are just the direct parents of the final, root meeting). With probability `p-required-resource-meeting-origin`, the meeting planner adds a new meeting as a parent of the current meeting, which initially contains only the taskforce member. The planner traverses to that meeting and checks the probability again. With probability 1−`p-required-resource-meeting-origin`, the current meeting becomes a resource meeting between the resource hat and the taskforce member. In a resource meeting, capability trades are planned to transfer the required capabilities to the taskforce members. This process is repeated until all of the resource hats have been assigned to taskforce members.

At this point, the meeting tree has all of the necessary meetings with trades to ensure that the taskforce will arrive at the task target with all of the required capabilities. The meeting planner then fills out the tree with additional meetings, participants, and capability trades. The additional meetings and trades are referred to as "decoys" because they are not directly involved in the task completion. The parameter `p-produce-decoy-meeting` is used to determine whether a decoy meeting should be added to a leaf meeting of the current meeting tree.

Once a meeting tree has been completely filled-out, it is added to a queue of current tasks and it will start to be executed at the next step of the simulation. During execution, the current leaves of each meeting tree are added to the `currently-executing-meetings` list and the Hats engine starts moving currently executing meeting participants toward their meeting locations. Once all of the meeting participants have arrived at a meeting location, the meeting lasts for two ticks, after which all hats not participating in more meetings are set "free" (and thus available to participate in new planned tasks). All other hats still reserved for meetings then begin moving to their next meeting.

The meeting trees created by this meeting planner typically have a depth ranging from 2 to 5. The frequency of tasks planned depends on both `p-start-new-task` and the number of hats in each organization (which comprise the resources available to the planner).

The Hats Simulator is designed to accommodate any meeting planner that adheres to a planner API. We are developing the API and anticipate using other planners. For example, a variation on the above planner would plan tasks that relate meetings as directed acyclic graphs (DAGs) as opposed to trees. This allows taskforce members to meet with one another repeatedly before the final meeting. We are also exploring other meeting topologies in conjunction with researchers in social network theory.

3 THE INFORMATION BROKER

Think of the Hats Simulator as a society in a box and your job, as an analyst, is to protect the society against terrorist taskforces. Specifically, you need to identify terrorist task forces as such before they damage beacons. To do so, you require information about the hats in the box. Information is acquired from an *Information Broker*, as shown previously in figure 1. The Information Broker will respond to questions from you, such as, *Where is* Hat_{27} *now?* and it will also provide information by subscription to analysts' tools (which in turn make requests for information). For example, a tool might issue a request like, *Identify everyone* Hat_{27} *meets in the next 100 steps*, or, *Tell me if* Hat_{27} *approaches a beacon with capabilities* c_1, c_7 *or* c_{29}.

Information comes at a price. Some is free, but information about states of the simulator that change over time is costly. The quality of the information obtained is determined by the amount paid. The following two sections describe the two central components to the request framework: the cost of information and noise. Together, these components make the Hats simulator an experimental environment in which to study the economics of the value of information in the task of identifying malevolent behavior in the Hats domain.

3.1 The cost of information

Three kinds of information are available from the Information Broker for free: (1) information about the population assumed to be available to the user (e.g., who the known terrorists are), (2) information about the Hats simulated world (e.g., the world-map dimensions, the list of beacons and their names, and the list of all of the capabilities that exist), and (3) some event bookkeeping (an event history, list of currently arrested hats, etc.). Information types 1 and 2 are determined

when the simulation is initialized and do not change over time; type 3 is updated at each step of the simulation.

For Information Broker requests that require payment, the amount paid (a real number) will determine a base probability, which in turn determines the accuracy of the requested information. In the current implementation, increased accuracy requires exponentially more "algorithmic dollars." The payment function, shown in Equation 1, maps payment to probability.

$$probability = 1 - \frac{1}{\log_2(\frac{payment}{5} + 2)} \qquad (1)$$

The same function is applied to every payment-based request.

3.2 Noise model

The development of a suitable noise model and the schemes for how noise is applied to requested information is, itself, an entire field of study. We list here three approaches, in increasing order of complexity:

1. The analyst may only request a particular piece of information once and must choose the level of payment for (and therefore quality of) the information at the time of request. No additional requests may be made. The analyst must decide at the time of request the value of that piece of information.
2. The analyst may request information multiple times. However, in order to receive information beyond previous request(s), the analyst must pay more than previous requests (according to the payment scale). Repeated requests at or below the same level will return precisely the same information, but paying more returns less noisy versions of the original request.
3. The analyst may request information multiple times, paying varying amounts. This approximates the existence of multiple information sources (for example, acquiring information from multiple witnesses of an event). Such multiple information sources might be made explicit, introducing the potential of modeling sources of trust relationships.

Many other schemes are possible, but these provide some indication of the wide variety of approaches to modeling noise.

The current implementation of the information broker employs the first scheme. The payment the analyst specifies determines the base probability p of whether, and to what degree, the information requested will be noisy: with probability p, the information requested is returned in its entirety, otherwise the noise model is applied.

Although the basic noise application scheme is simple, there still is a variety of types of information each of which requires a different noise model variant. The table in Figure 3 summarizes how different types of requested information are made noisy. Following the noise application scheme, analysts may only request each piece of information *once*. Some information, such as the capabilities currently carried by a hat (`ib-hat-capabilities`), is updated at each tick, so the analyst may request that information once each tick. Other information does not update, such as information about the members of a meeting that took place (`ib-meeting-participants`) – here the analyst is allowed only *one* request of this information. The column labeled "Request Frequency" shows the frequency with which an analyst may request information.

Request Return Type		IB Request	Request Frequency	Mean List Length … if information Exists	Mean List Length … if information ¬Exists	For Each Element: … if information Exists	For Each Element: … if information ¬Exists
List	hats	ib-location-contents	Each tick	length of Original list	2	If *Noisy*: Select Random, Else… Select from Original Without Replacement	NIL
		ib-meeting-participants	1				
	capabilities	ib-hat-capabilities	Each tick				
	trades	ib-meeting-trades	1		3		
	times	ib-meeting-times	Each tick	3	3	Select without replacement from most recent; NIL if none left	NIL
Element	time	ib-hat-obituary	Each tick			If *Noisy*: Exponential random with mean 1; If 0, then NIL	
	location	ib-hat-location	Each tick			If *Noisy*: Random location	
		ib-meeting-location	1				

Figure 3: Noise model

The table is split into two groups based on whether the requested information is a single element (bottom portion of the table) or a list of elements (top portion of the table).

3.2.1 Lists

Noise is applied to lists in two stages: first, noise affects the length of the list to be returned, and then noise is applied to each element of the list. The two main columns on the right-hand side of the list portion of the table indicate how noise is applied to list-length and to each element; in either case, noise is applied differently depending on whether or not the request is for information about entities that exist or events that occurred – true, non-noisy information about entities

that do not exist or events that did not occur is returned as `NIL`.

List length is determined by sampling a random value from a normal distribution with a standard deviation of 1.0 and a variable mean.[2] The "Mean List Length" column describes how the mean for the normal sampling distribution is set. For example, if the analyst requests the current contents of a Hats world location (using `ib-location-contents`), and there are in fact 3 hats at that location, then the length of the potential return information (3) determines the mean; subsequently, the noisy length of the list of hats that will be returned as a result of the request will be a random number selected from a normal distribution with mean 3, standard deviation 1. If, on the other hand, no hats exist at that location, then the mean of the normal distribution is 2 (as specified in Figure 3). These means have been chosen because they resulted in reasonable values during experimentation. If the analyst requests information involving a list and the selected random value rounds to 0 or lower, then the return value will be an empty list (or `NIL`).

Next, assignments are made for each element slot in the list to be returned. For each element, the noise model again uses the base probability p to determine whether the element slot will be noisy. If it is to be noisy, an element of the requested information type is uniformly randomly selected (with replacement) from the set of *all* elements of that type. For example, a random hat would be selected from *all* existing hats. In the case of trades, a noisy trade consists of two randomly chosen hats and one randomly chosen capability. With probability $1 - p$ the element will *not* be noisy. In this case, the element will be uniformly selected, *without* replacement, from the list of elements that would be returned if the information was uncorrupted; if the request is for information that does not exist, then that element of the list will be empty.

3.2.2 Elements

The elements portion of the table describes noise applied to information consisting of single elements. Random locations are selected when noise is applied to location information. A random location is chosen by selecting two random numbers, one for each coordinate component (x, y). The random numbers are selected from a standard normal distribution (mean 0, standard deviation 1.0). The value selected is then multiplied by the entire range of the x or y axis of the Hats World game board and divided by 10. This heuristic returns reasonable distances relative to the size of the game board dimensions. The adjusted value is then added to the true coordinate component. If the adjusted coordinates exceed the boarders of the game board, the amount exceeded is "reflected". For example, if a hat is at x-coordinate 3 and the adjustment is -5, then rather than returns an x-value of -2, the value is "reflected" to $x = 2$. If, on the other hand, the Game World maximum x size is 10 and the adjusted value is 12, then the value is "reflected" to $x = 8$.

[2]The sampled value is rounded to make it a valid list length.

4 ACTIONS

In addition to requesting information, the analyst playing the Hats game can also change a beacon's alert level and arrest hats. These actions affect the analyst's performance score (described in the next section).

4.1 Beacon Alerts

Each beacon can be set, by the analyst, to be in one of three alert levels: `off` (default), `low` or `high`, indicating no threat of an impending attack, a chance of an attack, and a likely attack, respectively. The Hats Simulator keeps track of beacon alert levels, including the amount of time a beacon alert is elevated (`low` or `high`) and whether actual attacks actually occur during elevated alerts. These statistics include counts of "hits" and "false positives," where "hits" $\equiv$ occurances of an attack while alert is elevated (above `off`), and "false-positives" $\equiv$ elevated alerts that begin and end with no beacon attack occurring. These scores are kept for both `low` and `high` alert levels. In general, the goal is to minimize the time beacon alerts are elevated, and `high` alerts are deemed "more costly" than `low` alerts. On the other hand, if an attack *does* occur on a beacon, it is generally better to have a higher alert level.

4.2 Arresting Hats

Analysts can also issue an arrest warrant for hats in order to prevent beacon attacks. A successful arrest results when the arrested hat is currently a member of terrorist taskforce. Arrests of any other hats, including hats that are terrorists but not currently part of a terrorist taskforce, result in arrest failure and are equivalent to a *false arrest* (a false positive). This is an important aspect of the semantics of "being a terrorist" in the Hats model: one can be a terrorist but not be guilty of any crime. Under this interpretation, "being a terrorist" is a matter of having a propensity to engage in terrorist acts. A terrorist act in the Hats domain is participating in an attack on a beacon. Thus, terrorist hats must be engaged in an ongoing terrorist activity to be successfully arrested. According to this model, if a hat previously

committed a terrorist act but is *not currently* part of a terrorist taskforce, it cannot be successfully arrested.

Successful arrests do not guarantee saving beacons. As noted, a beacon is only attacked when some subset of members from a taskforce carry the requisite capabilities that match the target beacon's vulnerabilities engage in a final meeting on said beacon. Thus, it is possible to successfully arrest a terrorist taskforce member but the other terrorist taskforce members still have the requisite capabilities to attack the beacon. If, on the other hand, the analyst successfully arrests a terrorist taskforce member carrying required capabilities that no other taskforce member carries, then the taskforce meeting will take place on the beacon, but it will not be attacked. This is counted as a "beacon save."

In the present version of Hats, the successful arrest of a hat does not remove it from the game – the hat will still behave as if it had not been arrested. It will still move toward goals and go to meetings. However, it will not be able to trade any of its capabilities nor contribute to enabling a beacon attack – it will be as though the hat were not present.

Currently, the statistics on beacon alert "hits," "false positives," "successful arrests," and "false arrests" are not combined into a uniform cost model. They are simply reported as additional measures of comparative player performance.

5 SCORING ANALYST PERFORMANCE

The Hats Simulator and Information Broker together provide an environment for testing analysts tools. Recall that the object of the game is to identify terrorist task forces before they damage beacons. Three kinds of costs are accrued:

- The cost of acquiring and processing information about a hat. This is the government in the bedroom or intrusiveness cost.
- The cost of falsely identifying benign hats as terrorist
- The cost of harm done by terrorists

The skill of analysts and the value of analysts tools can be measured in terms of these costs, and these are assessed automatically by the Hats simulator as the analyst plays the Hats game. The final report generated by the Hats Simulator after terminating a simulation run is divided up into four categories, as described in the following list:

- Costs: the total amount of "algorithmic dollars" spent on information from the Information Broker.
- Beacon Attacks: including the total number of terrorist attacks that succeeded and the total number of attacks that were stopped by successful arrests.
- Arrests: the number of successful arrests and the number of false-arrests (false-positives)
- Beacon Alerts: the number of low and high hits (the number of raised alerts during which an attack occurred), and the number of low and high false-positives (the number of raised alerts during which no attack occurred).

6 DISCUSSION

We are told by intelligence analysts that Hats has many attributes of "the real thing." Some say in the same breath that Hats ought to have other attributes, for instance, telephone communications, rapid transportation of hats around the board, different kinds of beacons, and so on. We resist these efforts to make Hats more "realistic" because for us, the purpose of Hats is to provide an enormously difficult detection problem with low domain knowledge overhead. No doubt Hats will change over time, but we will strive to keep it simple. Big, complex, covert, but simple. The other goal that guides our development of Hats is what we might call the "missing science" of intelligence analysis. To the best of our knowledge, in the current climate, analysts penalize misses more than false positives. This sort of utility function has consequences – raised national alert levels, lines at airports, and so on. Hats is intended to be a simulated world in which analysts can experiment with different utility functions. It is a laboratory in which scientific models of intelligence gathering, filtering, and use – models based on utility theory and information – can be tested and compared.

To meet these goals, our ongoing development of Hats includes the following: (1) increasing the scale and efficiency of the simulator to accommodate hundreds of thousands of hats running in reasonable time to conduct experiments and play in real-time; (2) building WebHats, a web-based interface to Hats, enabling any researcher with access to the web to make immediate use of Hats as a data source; (3) providing league tables of analyst/tool performance scores from playing the Hats game, promoting public competition to better intelligence analysis technology; and (4) developing a user-friendly interface to Hats, including more complex information querying and visual aids so that human analysts can play the Hats game more naturally.

7 HISTORY AND ACKNOWLEDGEMENTS

The Hats Simulator was conceived of by Paul Cohen and Niall Adams at Imperial College in the summer of 2002. Cohen implemented the first version of Hats, and David Westbrook, Clayton Morrison, Andrew Hannon and Michiharu Oshima have subsequently developed major portions of the simulator. Thanks also are due to Gary King for help. Bob Schrag at IET contributed useful ideas and built a simulator similar to Hats for DARPA's Evidence Extraction and Link Discovery (EELD) program. Work on this project was funded by EELD.

AUTHOR BIOGRAPHIES

PAUL R. COHEN is the deputy division director of the Intelligent Systems Division of the University of Souther California's Information Sciences Institute. In 2003 he became the Director of the Center for Research on Unexpected Events (CRUE). Dr. Cohen is currently on leave from the Department of Computer Science at the University of Massachusetts, where he has served for 20 years as a Professor and Director of the Experimental Knowledge Systems Laboratory. His PhD is from Stanford University in Computer Science and Psychology, in 1983. He served as a Councillor of the American Association for Artificial Intelligence, 1991–1994, and was elected in 1993 as a Fellow of the AAAI. His projects include AIID, an Architecture for the Interpretation of Intelligence Data; Capture the Flag, a wargaming environment; the Robot Baby project, in which a robot learns representations and their meanings sufficient for natural language and planning; and the Packrats project, in which rats are trained to carry video cameras for search-and-rescue operations. He also works on algorithms for finding patterns in temporal data. Dr. Cohen is interested in AI methodology, particularly empirical methods. His e-mail address is <cohen@isi.edu>, and his web page is http://eksl.cs.umass.edu/∼cohen/ .

CLAYTON T. MORRISON is a Postdoctoral Research Fellow in the Information Sciences Institute at the University of Southern California. Formerly, Dr. Morrison was a Senior Research Fellow in the Experimental Knowledge Systems Laboratory of the Computer Science Department at the University of Massachusetts. Dr. Morrison holds a Bachelors degree in Cognitive Science from Occidental College, and received his Masters and Ph.D. in Philosophy from Binghamton University. His research interests include the nature of representation and knowledge in humans and machines, cognitive development, and the rapid identification of unexpected behaviors in large populations. He is currently working on the development of a Bayesian blackboard system for the interpretation and analysis of asynchronous and noisy data from a variety of complex domains. His e-mail address is <clayton@isi.edu>, and his web page is http://eksl.cs.umass.edu/∼clayton/

Appendix C:

An Unsupervised Algorithm for Segmenting Categorical Timeseries into Episodes

Paul Cohen[1], Brent Heeringa[1], and Niall Adams[2]

[1] Department of Computer Science. University of Massachusetts, Amherst. Amherst, MA 01003
{cohen | heeringa}@cs.umass.edu
[2] Department of Mathematics. Imperial College. London, UK
n.adams@ic.ac.uk

Abstract. This paper describes an unsupervised algorithm for segmenting categorical time series into episodes. The VOTING-EXPERTS algorithm first collects statistics about the frequency and boundary entropy of ngrams, then passes a window over the series and has two "expert methods" decide where in the window boundaries should be drawn. The algorithm successfully segments text into words in four languages. The algorithm also segments time series of robot sensor data into subsequences that represent episodes in the life of the robot. We claim that VOTING-EXPERTS finds meaningful episodes in categorical time series because it exploits two statistical characteristics of meaningful episodes.

1 Introduction

Though we live in a continuous world, we have the impression that experience comprises episodes: writing a paragraph, having lunch, going for a walk, and so on. Episodes have hierarchical structure; for instance, writing a paragraph involves thinking of what to say, saying it, editing it; and these are themselves episodes. Do these examples of episodes have anything in common? Is there a domain-independent, formal notion of episode sufficient, say, for an agent to segment continuous experience into meaningful units?

One can distinguish three ways to identify episode boundaries: First, they may be *marked*, as spaces mark word boundaries and promoters mark coding regions in DNA. Second, episodes may be *recognized*. For instance, we recognize nine words in the sequence "itwasabrightcolddayinapriland". Third we might *infer* episode boundaries given the statistical structure of a series. For example, "juxbtbcsjhiudpmeebzjobqsjmboe" is formally (statistically) identical with "itwasabrightcolddayinapriland" — one is obtained from the other by replacing each letter with the adjacent one in the alphabet — however, the latter is easily segmented by recognition whereas the former requires inference.

This paper proposes two statistical characteristics of episode boundaries and reports experiments with an unsupervised algorithm called VOTING-EXPERTS based on these characteristics. We offer the conjecture that these characteristics are domain-independent and illustrate the point by segmenting text in four languages.

2 The Episode Boundary Problem

Suppose we remove all the spaces and punctuation from a text, can an algorithm figure out where the word boundaries should go? Here is the result of running VOTING-EXPERTS on the first 500 characters of George Orwell's *1984*. The ⋆ symbols are induced boundaries:

> Itwas ⋆ a ⋆ bright ⋆ cold ⋆ day ⋆ in ⋆ April ⋆ andthe ⋆ clockswere ⋆ st ⋆ ri ⋆ king ⋆ thi ⋆ rteen ⋆ Winston ⋆ Smith ⋆ his ⋆ chin ⋆ nuzzl ⋆ edinto ⋆ his ⋆ brea ⋆ st ⋆ in ⋆ aneffort ⋆ to ⋆ escape ⋆ the ⋆ vilewind ⋆ slipped ⋆ quickly ⋆ through ⋆ the ⋆ glass ⋆ door ⋆ sof ⋆ Victory ⋆ Mansions ⋆ though ⋆ not ⋆ quickly ⋆ en ⋆ ought ⋆ oprevent ⋆ aswirl ⋆ ofgrit ⋆ tydust ⋆ from ⋆ ent ⋆ er ⋆ inga ⋆ long ⋆ with ⋆ himThe ⋆ hall ⋆ ways ⋆ meltof ⋆ boiled ⋆ cabbage ⋆ and ⋆ old ⋆ ragmatsA ⋆ tone ⋆ endof ⋆ it ⋆ acoloured ⋆ poster ⋆ too ⋆ large ⋆ for ⋆ indoor ⋆ dis ⋆ play ⋆ hadbeen ⋆ tack ⋆ ed ⋆ tothe ⋆ wall ⋆ It ⋆ depicted ⋆ simplya ⋆ n ⋆ enormous ⋆ face ⋆ more ⋆ than ⋆ ametre ⋆ widethe ⋆ faceof ⋆ aman ⋆ of ⋆ about ⋆ fortyfive ⋆ witha ⋆ heavy ⋆ black ⋆ moustache ⋆ and ⋆ rugged ⋆ ly ⋆ handsome ⋆ featur

The segmentation is imperfect: Words are run together (*Itwas*, *aneffort*) and broken apart (*st* ⋆ *ri* ⋆ *king*). Occasionally, words are split between segments (*to* in *en* ⋆ *ought* ⋆ *oprevent*). Still, the segmentation is surprisingly good when one considers that it is based on nothing more than statistical features of subsequences of letters — not words, as no word boundaries are available — in Orwell's text.

How can an algorithm identify subsequences that are *meaningful* in a domain lacking any knowledge about the domain; and particularly, lacking positive and negative training instances of meaningful subsequences? VOTING-EXPERTS must somehow detect *domain-independent* indicators of the boundaries of meaningful subsequences. In fact, this is a good description of what it does. It implements a weak theory of domain-independent features of meaningful units. The first of these features is that entropy remains low inside meaningful units and increases at their boundaries; the second is that high-frequency subsequences are more apt to be meaningful than low-frequency ones.

3 Characteristics of Episodes

The features of episodes that we have implemented in the VOTING-EXPERTS algorithm are called *boundary entropy* and *frequency*:

Boundary entropy. Every unique subsequence is characterized by the distribution of subsequences that follow it; for example, the subsequence "en" in this sentence repeats seven times and is followed by tokens *c* (4 times), *t*, *s* and *"*, a distribution of symbols with an entropy value (1.66, as it happens). In general, every subsequence S has a boundary entropy, which is the entropy of the distribution of subsequences of length m that follow it. If S is an episode, then the boundary entropies of subsequences of S will have an interesting profile: They

will start relatively high, then sometimes drop, then peak at the last element of S. The reasons for this are first, that the predictability of elements within an episode increases as the episode extends over time; and second, the elements that immediately follow an episode are relatively uncertain. Said differently, within episodes, we know roughly what will happen, but at episode boundaries we become uncertain.

Frequency. Episodes, recall, are meaningful sequences. They are patterns in a domain that we call out as special, important, valuable, worth committing to memory, worth naming, etc. One reason to consider a pattern meaningful is that one can use it for something, like prediction. (Predictiveness is another characteristic of episodes nicely summarized by entropy.) Rare patterns are less useful than common ones simply because they arise infrequently, so all human and animal learning places a premium on frequency. In general, episodes are common patterns, but not all common patterns are episodes.

4 Related work

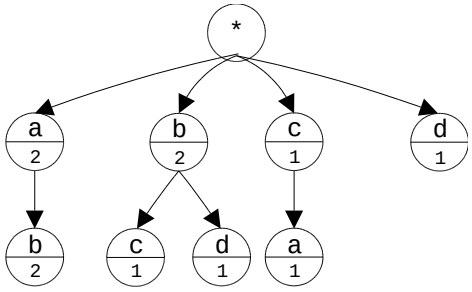
Many methods have been developed for segmenting time series. Of these, many deal with continuous time series, and are not directly applicable to the problem we are considering here. Some methods for categorical series are based on compression (e.g., [1]), but compression alone finds common, not necessarily meaningful, subsequences. Some methods are trained to find instances of patterns or templates (e.g., [2, 3]) or use a supervised form of compression (e.g., [4]), but we wanted an unsupervised method. There is some work on segmentation in the natural language and information retrieval literature, for instance, techniques for segmenting Chinese, which has no word boundaries in its orthography, but again, these methods are often supervised. The method in [5] is similar to ours, though it requires supervised training on very large corpora. The parsing based on mutual information statistics approach in [6] is similar to our notion of boundary entropy. [7] provides a developmentally plausible unsupervised algorithm for word segmentation, but the procedure assumes known utterance boundaries. [8] give an unsupervised segmentation procedure for Japanese, however it too supposes known sequence boundaries. With minor alterations, their segmentation technique is applicable to our domain, but we found that VOTING-EXPERTS consistently outperforms it. We know of no related research on characteristics of meaningful episodes, that is, statistical markers of boundaries of meaning-carrying subsequences.

5 The Voting Experts Algorithm

VOTING-EXPERTS includes experts that attend to boundary entropy and frequency and is easily extensible to include experts that attend to other characteristics of episodes. The algorithm simply moves a window across a time series and asks for each location in the window whether to "cut" the series at that location. Each expert casts a vote. Each location takes n steps to traverse a

window of size n, and is seen by the experts in n different contexts, and may accrue up to n votes from each expert. Given the results of voting, it is a simple matter to cut the series at locations with high vote counts. Here are the steps of the algorithm:

Build an ngram trie of depth $n+1$. Nodes at level $i+1$ of the trie represent ngrams of length i. The children of a node are the extensions of the ngram represented by the node. For example, $a\ b\ c\ a\ b\ d$ produces the following trie of depth 3:



Every ngram of length 2 or less in the sequence $a\ b\ c\ a\ b\ d$ is represented by a node in this tree. The numbers in the lower half of the nodes represent the frequencies of the subsequences. For example, the subsequence ab occurs twice, and every occurrence of a is followed by b.

For the first 10,000 characters in Orwell's text, an ngram trie of depth 8 includes 33774 nodes, of which 9109 are leaf nodes. That is, there are over nine thousand unique subsequences of length 7 in this sample of text, although the average frequency of these subsequences is 1.1—most occur exactly once. The average frequencies of subsequences of length 1 to 7 are 384.4, 23.1, 3.9, 1.8, 1.3, 1.2, and 1.1.

Calculate boundary entropy. The boundary entropy of an ngram is the entropy of the distribution of tokens that can extend the ngram. The entropy of a distribution for a discrete random variable X is

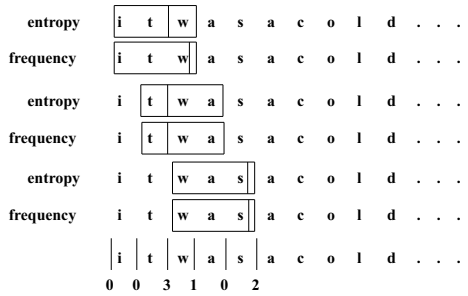
$$-\sum_{x \in X} p(x) \log p(x)$$

Boundary entropy is easily calculated from the trie. For example, the node a in the tree above has entropy equal to zero because it has only one child, ab, whereas the entropy of node b is 1.0 because it has two equiprobable children, bc and bd. Clearly, only the first n levels of the ngram tree of depth $n+1$ can have node entropy scores.

Standardize frequencies and boundary entropies. In most domains, there is a systematic relationship between the length and frequency of patterns; in general, short patterns are more common than long ones (e.g., on average, for subsets of 10,000 characters from Orwell's text, 64 of the 100 most frequent patterns are of length 2; 23 are of length 3, and so on). Our algorithm will compare the frequencies and boundary entropies of ngrams of different lengths, but in all cases we will be comparing how *unusual* these frequencies and entropies

are, relative to other ngrams of the same length. To illustrate, consider the words "a" and "an." In the first 10000 characters of Orwell's text, "a" occurs 743 times, "an" 124 times, but "a" occurs only a little more frequently than other one-letter ngrams, whereas "an" occurs much more often than other two-letter ngrams. In this sense, "a" is ordinary, "an" is unusual. Although "a" is much more common than "an" it is much less unusual relative to other ngrams of the same length. To capture this notion, we standardize the frequencies and boundary entropies of the ngrams. To standardize a value in a sample, subtract the sample mean from the value and divide by the sample standard deviation. This has the effect of expressing the value as the number of standard deviations it is away from the sample mean. Standardized, the frequency of "a" is 1.1, whereas the frequency of "an" is 20.4. In other words, the frequency of "an" is 20.4 standard deviations above the mean frequency for sequences of the same length. We standardize boundary entropies in the same way, and for the same reason.

Score potential segment boundaries. In a sequence of length k there are $k-1$ places to draw boundaries between segments, and, thus, there are 2^{k-1} ways to divide the sequence into segments. Our algorithm is greedy in the sense that it considers just $k-1$, not 2^{k-1}, ways to divide the sequence. It considers each possible boundary in order, starting at the beginning of the sequence. The algorithm passes a window of length n over the sequence, halting at each possible boundary. All of the locations within the window are considered, and each garners zero or one vote from each expert. Because we have two experts, for boundary-entropy and frequency, respectively, each possible boundary may accrue a maximum of $2n$ votes. This is illustrated below.



A window of length 3 is passed along the sequence `itwasacold`. Initially, the window covers `itw`. The entropy and frequency experts each decide where they could best insert a boundary within the window (more on this, below). The entropy expert favors the boundary between `t` and `w`, while the frequency expert favors the boundary between `w` and whatever comes next. Then the window moves one location to the right and the process repeats. This time, both experts decide to place the boundary between `t` and `w`. The window moves again and both experts decide to place the boundary after `s`, the last token in the window. Note that each potential boundary location (e.g., between `t` and `w`) is seen n times for a window of size n, but it is considered in a slightly different context each time the window moves. The first time the experts consider the boundary

between `w` and `a`, they are looking at the window `itw`, and the last time, they are looking at `was`. In this way, each boundary gets up to $2n$ votes, or $n = 3$ votes from each of two experts. The `wa` boundary gets one vote, the `tw` boundary, three votes, and the `sa` boundary, two votes.

The experts use slightly different methods to evaluate boundaries and assign votes. Consider the window `itw` from the viewpoint of the boundary entropy expert. Each location in the window bounds an ngram to the left of the location; the ngrams are `i`, `it`, and `itw`, respectively. Each ngram has a standardized boundary entropy. The boundary entropy expert votes for the location that produces the ngram with the highest standardized boundary entropy. As it happens, for the ngram tree produced from Orwell's text, the standardized boundary entropies for `i`, `it`, and `itw` are 0.2, 1.39 and 0.02, so the boundary entropy expert opts to put a boundary after the ngram `it`.

The frequency expert places a boundary so as to maximize the sum of the standardized frequencies of the ngrams to the left and the right of the boundary. Consider the window `itw` again. If the boundary is placed after `i`, then (for Orwell's text) the standardized frequencies of `i` and `tw` sum to 1.73; if the boundary is placed after `it`, then the standardized frequencies of `it` and `w` sum to 2.9; finally, if it is placed after `itw`, the algorithm has only the standardized frequency of `itw` to work with; it is 4.0. Thus, the frequency expert opts to put a boundary after `itw`.

Segment the sequence. Each potential boundary in a sequence accrues votes, as described above, and now we must evaluate the boundaries in terms of the votes and decide where to segment the sequence. Our method is a familiar "zero crossing" rule: If a potential boundary has a locally maximum number of votes, split the sequence at that boundary. In the example above, this rule causes the sequence `itwasacold` to be split after `it` and `was`. We confess to one embellishment on the rule: The number of votes for a boundary must exceed an absolute threshold, as well as be a local maximum. We found that the algorithm splits too often without this qualification.

Let us review the design of the experts and the segmentation rule, to see how they test the characteristics of episodes described earlier. The boundary entropy expert assigns votes to locations where the boundary entropy peaks locally, implementing the idea that entropy increases at episode boundaries. The frequency expert tries to find a "maximum likelihood tiling" of the sequence, a placement of boundaries that makes the ngrams to the left and right of the boundary as likely as possible. When both experts vote for a boundary, and especially when they vote repeatedly for the same boundary, it is likely to get a locally-maximum number of votes, and the algorithm is apt to split the sequence at that location.

6 Evaluation

In these experiments, induced boundaries stand in six relationships to episodes.

1. The boundaries coincide with the beginning and end of the episode;

2. The episode falls entirely within the boundaries and begins or ends at one boundary.
3. The episode falls entirely within the boundaries but neither the beginning nor the end of the episode correspond to a boundary.
4. One or more boundaries splits an episode, but the beginning and end of the episode coincide with boundaries.
5. Like case 4, in that boundaries split an episode, but only one end of the episode coincides with a boundary.
6. The episode is split by one or more boundaries and neither end of the episode coincides with a boundary.

These relationships are illustrated graphically in Figure 1, following the convention that horizontal lines denote actual episodes, and vertical lines denote induced boundaries. The cases can be divided into three groups. In cases 1 and 4, boundaries correspond to both ends of the episode; in cases 2 and 5, they correspond to one end of the episode; and in cases 3 and 6, they correspond to neither end. We call these cases *exact*, *dangling*, and *lost* to evoke the idea of episodes located exactly, dangling from a single boundary, or lost in the region between boundaries.

We use both hit and false-positive rates to measure the accuracy of our episode finding algorithms. To better explain the trade-offs between hits and false-positives we employ the F-measure [9]. This standard comparison metric finds the harmonic mean between precision and recall is defined as

$$\text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where Recall is the hit-rate and Precision is the ratio of correct hits to proposed hits. Note that the difference in proposed and correct hits yields the number of false positives. Higher F-measures indicate better overall performance.

For control purposes we compare VOTING-EXPERTS with two naive algorithms. The first generates a random, sorted sequence of boundaries that is

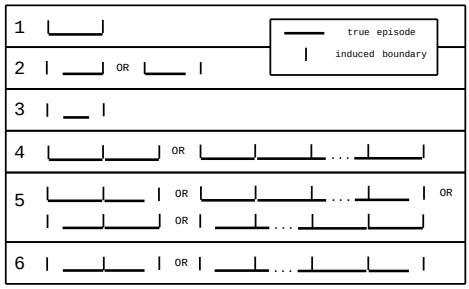


Fig. 1. A graphical depiction of the relationships between boundaries and episodes. Horizontal lines denote true episodes; their ends the correct boundaries. Vertical lines denote induced episode boundaries.

equal in size to the actual number of episodes. We call this algorithm RANDOM-SAMPLE. The second algorithm induces a boundary at every location. We call this algorithm ALL-LOCATIONS.

In many of these experiments, we compare the results of VOTING-EXPERTS with another unsupervised algorithm, SEQUITUR, which also finds structure in categorical time series. SEQUITUR is a compression-based algorithm that builds a context-free grammar from a string of discrete tokens [1]. It has successfully identified structure in both text and music. This structure is denoted by the rules of the induced grammar. Expanding the rules reveals boundary information. In our experiments, expanding only the rule associated with the start symbol – what we refer to as level 1 expansion – most often gives the highest F-measure.

6.1 Word Boundaries

We removed spaces and punctuation from texts in four languages and assessed how well VOTING-EXPERTS could induce word boundaries. We take word boundaries as our gold standard for meaning-carrying units in text because they provide, in most cases, the most unambiguous and uncontentious denotation of episodes. Clearly word prefixes and suffixes might also carrying meaning, but most humans would likely segment a discrete stream of text into words.

Algorithm	F-measure	Hit Rate	F.P. Rate	Exact %	Dangling %	Lost %
VOTING-EXPERTS	.76	.80	.27	.63	.34	.03
SEQUITUR	.58	.58	.43	.30	.56	.14
ALL-LOCATIONS	.36	1.0	.78	1.0	0.0	0.0
RANDOM-SAMPLE	.21	.22	.79	.05	.34	.61

Table 1. Results of running four different algorithms on George Orwell's *1984*.

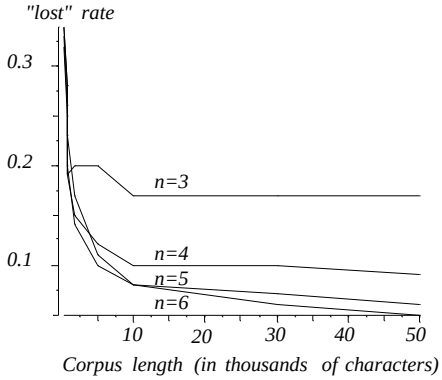
English We ran VOTING-EXPERTS, SEQUITUR, and both naive algorithms on the first 50,000 characters of Orwell's *1984*. The detailed results are given in Table 1. VOTING-EXPERTS performed best when the window length was 7 and the threshold 4. The algorithm induced 12153 boundaries, for a mean episode length of 4.11. The mean word length in the text was 4.49. The algorithm induced boundaries at 80% of the true word boundaries (the hit rate) missing 20% of the word boundaries. 27% of the induced boundaries did not correspond to word boundaries (the false positive rate). Exact cases, described above, constitute 62.6% of all cases; that is, 62.6% of the words were bounded at both ends by induced boundaries. Dangling and lost cases constitute 33.9% and 3.3% of all cases, respectively. Said differently, only 3.3% of all words in the text got lost between episode boundaries. These tend to be short words, in fact, 59% of the lost words have length 3 or shorter and 85% have length 5 or shorter. In contrast,

all 89% of the words for which the algorithm found exact boundaries are of length 3 or longer.

SEQUITUR performed best when expanding only to the level 1 boundaries. That is, it achieved its highest F-measure by not further expanding any non-terminals off the sentential production. Expanding to further levels leads to a substantial increase in the false positive rate and hence the overall decrease in F-measure. For example, when expanding to level 5, SEQUITUR identified 78% of the word boundaries correctly, 20% dangling and only 2% missed. This happens because it is inducing more boundaries. In fact, at level 5, the false-positive rate of 68% is near the 78% maximum false positive rate achieved by ALL-LOCATIONS. The same behavior occurs to a smaller extent in VOTING-EXPERTS when the splitting threshold is decreased. For example, with a window length of 4 and a threshold of 2, VOTING-EXPERTS finds 74% of the word boundaries exactly but the F-measure decreases because a corresponding increase in the false-positive rate. In general, SEQUITUR found likely patterns, but these patterns did not always correspond to word boundaries.

It is easy to ensure that all word boundaries are found, and no word is lost: use ALL-LOCATIONS to induce a boundary between each letter. However, this strategy induces a mean episode length of 1.0, much shorter than the mean word length. The false-positive count equals the total number of non-boundaries in the text and the false-positive rate converges to the ratio of non-boundaries to total locations (.78). In contrast, VOTING-EXPERTS finds roughly the same number of episodes as there are words in the text and loses very few words between boundaries. This success is evident in the high F-measure (.76) achieved by VOTING-EXPERTS. Not surprisingly, RANDOM-SAMPLE performed poorest on the text.

The appropriate control conditions for this experiment were run and yielded the expected results: VOTING-EXPERTS performs marginally less well when it is required to segment text it has not seen. For example, if the first 10,000 characters of Orwell's text are used to build the ngram tree, and then the algorithm is required to segment the next 10,000 characters, there is a very slight decrement in performance. The algorithm performs very poorly given texts of random words, that is, subsequences of random letters. The effects of the corpus size and the window length are shown in the following graph. The proportion of "lost" words (cases 3 and 6, above) is plotted on the vertical axis, and the corpus length is plotted on the horizontal axis. Each curve in the graph corresponds to a window length, k. The proportion of lost words becomes roughly constant for corpora of length 10,000 and higher.



Said differently, corpora of this length seem to be required for the algorithm to estimate boundary entropies and frequencies accurately. As to window length, recall that a window of length n means each potential boundary is considered n times by each expert, in n different contexts. Clearly, it helps to increase the window size, but the benefit diminishes.

Further evidence of VOTING-EXPERTS ability to find meaningful word boundaries is given in Figures 2 and 3. In Figure 2 we graph the percentage of exact word matches as a function of word length. For example, SEQUITUR exactly matches 30% of words having length 15 while VOTING-EXPERTS matches 70%. The curves converge at word length 17 because only two words in our corpus have length 17 and both algorithms find only one of them. The curves roughly mimic each other except in the word length interval from 2 to 4. In this period, VOTING-EXPERTS accelerates over SEQUITUR because it finds disproportionately more exact matches than SEQUITUR. This phenomenon is even easier to see in Figure 3. Here cumulative percentage of exact word matches is plotted as a function of word lengths and the distribution of word lengths is given behind the curves. The slope of VOTING-EXPERTS is steeper than SEQUITUR in the interval from 2 to 4 revealing the success it has on the most frequent word lengths. Furthermore, words with length 2, 3, and 4 comprise over 57% of the Orwell corpus, so at places where accuracy is perhaps most important, VOTING-EXPERTS performs well.

Chinese, German and Roma-ji As a test of the generality of VOTING-EXPERTS, we ran it on corpora of Roma-ji, Chinese and German texts. Roma-ji is a transliteration of Japanese into roman characters. The Roma-ji corpus was a set of Anime lyrics comprising 19163 characters. The Chinese text comes from Guo Jim's Mandarin Chinese PH corpus. The PH corpus is taken from stories in newspaper texts and is encoded in in the standard GB-scheme. Franz Kafka's *The Castle* in the original German comprised the final text. For comparison purposes we selected the first 19163 characters of Kafka's text and the same number of characters from *1984* and the PH corpus. As always, we stripped away spaces and punctuation, and the algorithm induced word boundaries. The window length was 6. The results are given in Table 2.

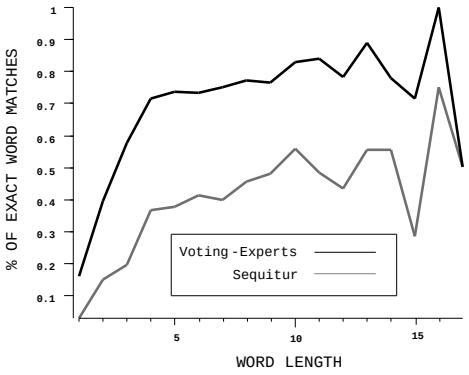


Fig. 2. A comparison of exact match-rate on a per-word basis between SEQUITUR and VOTING-EXPERTS.

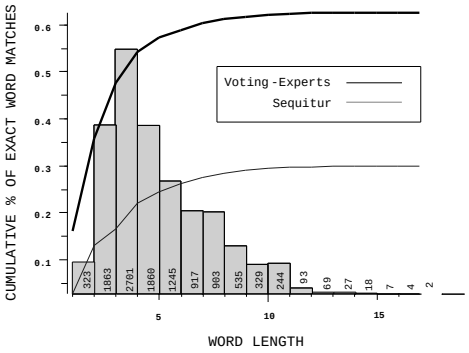


Fig. 3. A comparison of cumulative exact match-rate over word length for SEQUITUR and VOTING-EXPERTS. The background histogram depicts the distribution of word lengths in the Orwell corpus.

Clearly the algorithm is not biased to do well on English. In particular, it performs very well on Kafka's text, losing only 4% of the words and identifying 61% exactly. The algorithm performs less well with the Roma-ji text; it identifies fewer boundaries accurately (i.e., places 34% of its boundaries within words) and identifies fewer words exactly. VOTING-EXPERTS performed worst on Chinese corpus. Only 42% of the boundaries were identified although the false positive rate is an extremely low 7%. The explanation for these results has to do with the lengths of words in the corpora. We know that the algorithm loses disproportionately many short words. Words of length 2 make up 39% of the Chinese corpus, 32% of the Roma-ji corpus, 17% of the Orwell corpus, and 10% of the Kafka corpus, so it is not surprising that the algorithm performs worst on the Chinese corpus and best on the Kafka corpus.

VOTING-EXPERTS	F-measure	Hit Rate	F.P. Rate	Exact %	Dangling %	Lost %
German	.75	.79	.31	.61	.25	.04
English	.71	.76	.33	.58	.38	.04
Roma-ji	.65	.64	.34	.37	.53	.10
Chinese	.57	.42	.07	.13	.57	.30

Table 2. Results of running VOTING-EXPERTS on Franz Kafka's *The Castle*, Orwell's 1984, a subset of the Chinese PH corpus of newspaper stories, and a set of Roma-ji Anime lyrics.

If we incorporate the knowledge that Chinese words are rather short in length by decreasing the splitting threshold, we can increase the F-measure of VOTING-EXPERTS to 77% on the PH corpus. In general, knowledge of the mean episode length can help improve the boundary detection of VOTING-EXPERTS. Like [8], pretraining on a small amount of segmented text may be sufficient to find suitable window and threshold values.

6.2 Robot Episodes

We ran VOTING-EXPERTS and SEQUITUR on a multivariate timeseries of robot controller data comprising 17788 time steps and 65 unique states. Each state was mapped to a unique identifier, and these tokens were given to the algorithm as input. The timeseries data was collected with a Pioneer 2 mobile robot, equipped with sonar and a Sony pan-tilt-zoom camera. The robot wandered around a room-size playpen for 30 minutes looking for interesting objects. Upon finding an object, the robot orbited it for a few minutes. The multivariate timeseries consisted of eight binary variables representing different controllers in our agent architecture. Each variable is 1 when its corresponding controller is active and 0 when its inactive, so potentially, we have $2^8 = 256$ different states, but as mentioned earlier, only 65 manifested during the experiment.

- MOVE-FORWARD
- TURN
- COLLISION-AVOIDANCE
- VIEW-INTERESTING-OBJECT
- RELOCATE-INTERSTING-OBJECT
- SEEK-INTERESTING-OBJECT
- CENTER-CHASIS-ON-OBJECT
- CENTER-CAMERA-ON-OBJECT

This timeseries can be broken up into five different observable robot behaviors. Each behavior represents a qualitatively different episode in the timeseries. We denote these episodes as

- FLEEING
- WANDERING

– AVOIDING
– ORBITING-OBJECT
– APPROACHING-OBJECT

Table 3 summarizes the results of running VOTING-EXPERTS and SEQUITUR on the robot controller data. The definition of hit-rate and false-positive rate is slightly different here. Because the controller data can be noisy at the episode boundaries, we allow *hits* a window of length 1 in either temporal direction. For example, if we induce a boundary at location 10, but the actual boundary is at location 9, we still count it as a hit. We also enforce a rule that actual boundaries can only count once toward induced boundaries. For example, if we induce a boundary at 8 and count it as a hit toward the actual boundary 9, the induced boundary at 10 can no longer count toward 9.

The mean episode length in the robot controller data is 7.13. This length is somewhat smaller than expected because the robot often gets caught up in the corners of its playpen for periods of time and performs a series of wandering, avoiding, and fleeing behaviors to escape. The total number of true episodes was 2491. VOTING-EXPERTS induced 3038 episodes with a hit rate of 66% and a false-positive rate of 46% for a combined F-measure of 59%. Like on Orwell, VOTING-EXPERTS consistently outperforms SEQUITUR on the F-measure. SEQUITUR does best when expanding to the level 1 boundaries. The transition from level 1 to level 2 produces a sharp increase in the false-positive rate with a corresponding increase in hit rate, however the F-measure decreases slightly. At level 5, SEQUITUR loses only 8% of the episodes but its false-positive rate is 78%, which is near the maximum possible rate of 86%.

Robot Data	F-measure	Hit Rate	F.P. Rate	Exact %	Dangling Rate	Lost Rate
SEQUITUR						
Level 1	.55	.57	.47	.17	.37	.46
Level 2	.51	.77	.62	.34	.37	.29
Level 3	.32	.88	.71	.48	.33	.19
Level 4	.38	.94	.76	.56	.32	.12
Level 5	.36	.97	.78	.63	.29	.08
VOTING-EXPERTS						
Depth 7, Threshold 4	.59	.66	.46	.20	.39	.41
Depth 9, Threshold 6	.59	.60	.41	.18	.38	.44
Depth 5, Threshold 2	.56	.80	.56	.27	.42	.31

Table 3. Results of running SEQUITUR and VOTING-EXPERTS on 30 minutes of robot controller data.

7 Conclusion

For an agent to generalize its experiences, it must divide them into meaningful units. The VOTING-EXPERTS algorithm uses statistical properties of categorical time series to segment them into episodes without supervision or prior training. Although the algorithm does not use explicit knowledge of words or robot behaviors, it detects episodes in these domains. The algorithm successfully segments texts into words in four languages. With less success, VOTING-EXPERTS segments robot controller data into activities. In the future we will examine how other, domain-independent experts can help improve performance. Additionally we are interested in unifying the frequency and boundary entropy experts to more accurately capture the balance of strengths and weaknesses of each method. On a related note, we could employ supervised learning techniques to learn a weigh parameter for the experts, however we favor the unification approach because it removes a parameter from the algorithm and keeps the method completely unsupervised, The idea that meaningful subsequences differ from meaningless ones in some formal characteristics—that syntactic criteria might help us identify semantic units—has practical as well as philosophical implications.

8 Acknowledgments

We are grateful to Ms. Sara Nishi for collecting the corpus of Anime lyrics. This research is supported by DARPA under contract numbers DARPA/USASMDCDASG60-99-C-0074 and DARPA/AFRLF30602-01-2-0580. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of DARPA or the U.S. Government.

References

1. Nevill-Manning, C.G., Witten, I.H.: Identifying hierarchical structure in sequences: A linear-time algorithm. Journal of Artificial Intelligence Research **7** (1997) 67–82
2. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery **1** (1997) 259–289
3. Garofalakis, M.N., Rastogi, R., Shim, K.: SPIRIT: Sequential pattern mining with regular expression constraints. In: The VLDB Journal. (1999) 223–234
4. Teahan, W.J., Wen, Y., McNab, R.J., Witten, I.H.: A compression-based algorithm for chinese word segmentation. Computational Linguistics **26** (2000) 375–393
5. Weiss, G.M., Hirsh, H.: Learning to predict rare events in event sequences. In: Knowledge Discovery and Data Mining. (1998) 359–363
6. Magerman, D., Marcus, M.: Parsing a natural language using mutual information statistics. In: Proceedings, Eighth National Conference on Artificial Intelligence (AAAI 90). (1990) 984–989

7. Brent, M.R.: An efficient, probabilistically sound algorithm for segmentation and word discovery. Machine Learning **45** (1999) 71–105
8. Ando, R.K., Lee, L.: Mostly-unsupervised statistical segmentation of japanese: Application to kanji. In: Proceedings of North American Association for Computational Linguistics (NAACL). (2000) 241–248
9. Van Rijsbergen, C.J.: Information Retrieval, 2nd edition. Dept. of Computer Science, University of Glasgow (1979)

Appendix D:

Simulating Terrorist Threat in The Hats Simulator

Clayton T. Morrison[1], *Paul R. Cohen*[1], *Gary W. King*[2], *Joshua Moody*[1] *and Andrew Hannon*[2]

[1] USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292, USA
{clayton,cohen,moody}@isi.edu

[2] University of Massachusetts
140 Governors Drive
Amherst, MA 01003, USA
{gwking,hannon}@isi.edu

Keywords: Effects Based Nodal Analysis, Multiple Competing Hypotheses, Terrorism, Simulation

Abstract

The Hats Simulator is a lightweight proxy for many intelligence analysis problems, and thus a test environment for analysts' tools. It is a virtual world in which many agents engage in individual and collective activities. Most agents are benign, some intend harm. Agent activities are planned by a generative planner. Playing against the simulator, the job of the analyst is to find harmful agents before they carry out attacks. The simulator maintains information about all agents. However, information is hidden from the analyst and some is expensive. After each game, the analyst is assessed a set of scores including the cost of acquiring information about agents, the cost of falsely accusing benign agents, and the cost of failing to detect harmful agents. The simulator is implemented and currently manages the activities of up to a hundred thousand agents.

1. Introduction

The Hats Simulator was designed originally to meet the needs of academic researchers who want to contribute technology to Homeland Security efforts but lack access to domain experts and classified problems. Most academic researchers do not have security clearances and cannot work on real data, yet they want to develop tools to help analysts. In any case, real data sets are expensive: They cost a lot to develop from scratch or by "sanitizing" classified data. They also are domain-specific, yet much of the domain expertise is classified. Because data sets are expensive, many that have been made available to researchers are relatively small and the patterns to be detected within them are fixed, few, and known, so working with these data sets is a bit like solving a single "Where's Waldo" puzzle. Sometimes there also is the problem that real data sets model "signal" (terrorist activities) not "noise" (everything else) yet extracting signal from noise is a great challenge. Data sets in general are static, whereas data become available to analysts over time. It would be helpful to have a data *feed*, something that generates data as events happen. To validate analysts' tools, it would be helpful to have a generator of terrorist and non-terrorist activities. The generator should be parameterized for experimental purposes (e.g., varying the distinctiveness of terrorist activities, to make them more or less easily recognizable); and it should come up with novel activities, requiring analysts and their tools to both recognize known patterns and reason about suspicious patterns.

Hats is home to hundreds of thousands of agents (hats) which travel to meetings. Some hats are covert terrorists and a very few hats are known terrorists. All hats are governed by plans generated by a planner. Terrorist plans end in the destruction of landmarks. The object of a game in the Hats simulator is to find terrorist task forces before they carry out their attacks. One pays for information about hats, and also for false arrests and destroyed landmarks. At the end of a game, one is given a score, which is the sum of these costs. The goal is to play Hats rationally, that is, to catch terrorist groups with the least combined cost of information, false arrests, and destroyed landmarks. Thus Hats serves as a test bed not only for analysts' tools but also for new theories of rational intelligence analysis. Hats encourages players to ask only for the information they need, and to not accuse hats or issue alerts without justification.

The Hats simulator is very lightweight: Agents have few attributes and engage in few elementary behaviors; however, the number of agents is enormous, and plans can involve simultaneously many agents and a great many instances of behaviors. The emphasis in Hats is not domain knowledge but managing enormous numbers of hypotheses based on scant, often inaccurate information. By simplifying agents and their elementary behaviors, we de-emphasize the domain knowledge required to identify terrorist threats and emphasize covertness, complex group behaviors over time, and the frighteningly low signal to noise ratio.

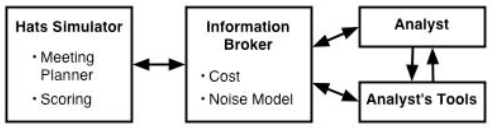


Figure 1 – Information Broker Interface to the Hats Simulator

The Hats environment consists of the core simulator and an information broker. The information broker is responsible for handling requests for information about the state of the simulator and thus forms the interface between the simulator and the analyst and her tools (see Figure 1). Some information has a cost, and the quality of information returned is a function of the "algorithmic dollars" spent. Analysts may also take actions: they may raise beacon alerts in an attempt to anticipate an attack on a beacon, and they may arrest agents believed to be planning an attack. Together, information requests and actions form the basis of scoring analyst performance in identifying terrorist threats and preventing terrorist attacks. Scoring is assessed automatically and serves as the basis for analytic comparison between different analysts and tools. The simulator is implemented and manages the activities of up to a hundred thousand agents.

The following sections outline the Hats domain, including how we generate populations of hats and how the planner schedules hat meetings. We describe the information request framework, the actions the analyst may take, and scoring. We conclude with a discussion of the future of the Hats Simulator.

2. The Hats Domain

The Hats Simulator models a "society in a box" consisting of many very simple agents, hereafter referred to as *hats*. Hats get its name from the classic spaghetti western, in which heroes and villains are identifiable by the colors of their hats. The Hats society also has its heroes and villains, but the challenge is to identify which color hat they should be wearing, based on how they behave. Some hats are known terrorists; others are *covert* and must be identified and distinguished from the *benign* hats in the society.

Hats is staged in a two-dimensional grid on which hats move around, go to meetings and trade capabilities. The grid consists of two kinds of locations: those that have no value, and high-valued locations called *beacons* that terrorists would like to attack. All beacons have a set of attributes, or *vulnerabilities*, corresponding to the *capabilities* which hats carry. To destroy a beacon, a task force of terrorist hats must possess capabilities that match the beacon's vulnerabilities, as a key matches a lock. In general, these capabilities are not unique to terrorists, so one cannot identify terrorist hats only on the basis of their capabilities.

The Hats society is structured by organizations. All hats belong to at least two organizations and some hats belong to many. Terrorist organizations host only known and covert terrorist hats. Benign organizations, on the other hand, may contain any kind of hat, including known and covert terrorists.

2.1 Population Generation

Hats populations may be built by hand or generated by the Hats Simulator. Because the constitution of a population affects the difficulty of identifying covert terrorists, population generation is parameterized. There are four sets of population parameters. The first set specifies the total number of known terrorists, covert terrorists and benign hats in the population. Another set defines the number of benign and terrorist organizations. Not all organizations have the same number of members, so a third set of parameters assigns the relative numbers of hats that are members of each organization, represented as a ratio among organizations. For example, the ratio 2:1:1 means that the first organization has twice as many members as the other two. Finally, hats may be members of two or more organizations. An overlap parameter determines the percentage of hats in each organization that are members of two or more *other* organizations. Since hat behaviors are governed by their organization membership, as we will see in the next section, organization overlap affects how difficult it is to identify covert terrorist hats. To generate populations with hundreds of thousands of hats and thousands of organizations, we use a randomized algorithm that estimates organization overlap percentage and membership ratios while matching the total number of organizations and hats in the population. When the population is generated, each hat is assigned a *native* capability that they will carry throughout the duration of the simulation, and a set of *traded* capabilities that are temporary, expiring after some number of ticks (e.g., within 40 ticks). Hats are also assigned random locations in the Hats grid world.

2.2 Meeting Generation

Hats act individually and collectively, but always planfully. In fact, the actions of hats are planned by a generative planner. Benign hats congregate at locations including beacons. Terrorist hats meet, acquire capabilities, form task forces, and attack beacons. The purpose of the planner is to construct an elaborate "shell game" in which capabilities are passed among hats in a potentially long sequence of meetings, culminating in a final meeting at a target. By moving capabilities among hats, the planner masks its intentions. Rather than directing half a dozen hats with capabilities required for an attack to march purposefully up to a beacon, instead hats with required capabilities pass them on to other hats, and eventually a capable task force appears at the beacon.

Each organization has a generative planner that plans tasks for its members. Hats that are currently participating in a task are *reserved*; hats not currently part of a task are *free*. At each tick, each organization has a chance of begin-

ning a new task. When a new task is started, the Hats meeting planner creates a *task force*, a subset of hats selected from the free hats of the organization. The size of a task-force is controlled by a parameter. The planner next selects a *target location* in the Hats world. With some probability, that location may be a beacon, otherwise a random location is selected. If a beacon is selected as the target, the goal of the task is to bring to that location the set of *required capabilities* that match the vulnerabilities of the beacon. If the location is not a beacon, a random set of required capabilities is selected as the set to bring to the location.

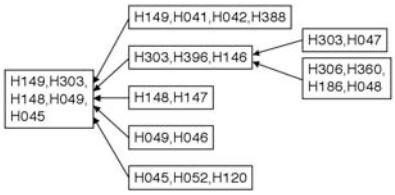


Figure 2 – Example of a generated meeting tree. Each box represents a meeting and contains a list of participating hats. Arrows indicate the temporal order of meetings.

Task force members may or may not already possess the required capabilities; usually they don't. The planner creates a set of meetings designed to ensure that the task force acquires all of the required capabilities before it reaches the target location. This is accomplished by constructing a *meeting tree* that specifies meetings and their temporal order. Figure 2 shows an example meeting tree, where boxes represent planned meetings among hats and arrows represent the planned temporal partial order of meetings. The tree is "inverted" in the sense that the arrows point from leaves inward toward the root of the tree. Parent meetings, where arrows originate, are executed first. When all of the parent meetings of a child meeting have completed, then the child meeting happens. This ensures that none of the hats that participate in the child meeting are busy in other meetings. Meeting execution means that the hats participating in the meeting begin moving toward the meeting location. The final, root meeting takes place at the task target location and includes all of the task force hats. The locations of the other meetings in the tree are selected randomly.

Initially, the meeting tree is skeletal, containing meetings whose only participants are the task force members themselves. From the organization's remaining free hats, the planner selects a second group of *resource* hats that carry required capabilities not currently carried by the task force. Resource hats are randomly assigned to existing meetings, and trades of required capabilities are scheduled to take place during the meeting. The planner finishes tree construction by adding decoy meetings, spurious trades and additional free hats not already involved in moving required capabilities to the goal. A constraint maintained throughout tree construction is that at least one hat from a parent meeting will go on to meet in a child meeting. These hats will either be task force members, resource hats carrying required capabilities to trade in the next meeting, or will be decoy hats arriving from decoy meetings.

Completed meeting trees are added to a queue of pending tasks. At each tick, the simulator engine searches the task queue for meetings with no currently executing parent meetings. These meetings are then assigned to a queue of currently executing meetings and the participant hats are incrementally moved toward the meeting location. When all of the participants have arrived at the meeting location, the meeting itself lasts for two ticks, after which all hats not participating in more meetings are set "free" and become available to participate in new meetings.

Meeting trees typically have a depth of 2 to 7. The frequency of new tasks depends on both the probability of starting a new task as well as the number of hats in each organization.

3. The Information Broker

We are currently developing a human interface to Hats to enable human analysts to play the Hats game. As an analyst playing the game, your job is to protect the Hats society from terrorist attacks. You need to identify terrorist task forces before they attack beacons, but you also need to avoid falsely accusing innocent hats. The only way to do this successfully is to gather information about hats, identify meetings, track capability trades and form hypotheses about the intentions of groups of hats. The *information broker* provides information about the state of the Hats world. The information broker will respond to questions such as *Where is Hat_{27} right now?* It will also provide information by subscription to analysts' tools, which in turn make information broker requests. For example, a tool might process requests like, *Identify everyone Hat_{27} meets in the next 100 ticks*, or, *Tell me if Hat_{27} approaches a beacon with capabilities c_1, c_7 or c_{29}.*

Some information is free, but information about states of the simulator that change over time is costly. The quality of the information obtained is determined by the amount paid. The following two sections describe the two central components of the request framework: the cost of information and noise. Together, these components make the Hats simulator an experimental environment for studying the economics of information value in the context of intelligence analysis.

3.1 The Cost of Information

Some information from the broker is free. This includes information about the population (who the known terrorists are), the simulator world (world-map dimensions), and some event bookkeeping (locations of attacks, a list of currently arrested hats). Other types of information require payment and the amount paid sets a base probability that is used to determine the accuracy of the information. Use of this base probability is explained in the next section. In the current implementation, increas-

ing accuracy requires exponentially more "algorithmic dollars." The function in Equation 1 maps payment to probability.

$$probability = 1 - \frac{1}{\log_2\left(\frac{payment}{5} + 2\right)} \quad (1)$$

The same function is applied to every payment-based request. This particular function was chosen because of its desirable rate of exponential growth, but other functions may be used.

3.2 Making Requested Information Noisy

Modeling noise is a topic suitable for an entire research program. There are many issues to consider, including whether one can request the same information multiple times, and if so, how information quality changes; whether one can get accurate information about events that occurred in the past; whether one can tell which information sources are reliable by asking several times the same question; and so on. We have started simply, imposing the constraint that the analyst may request a particular piece of information only once. This means that they must select the level of payment for the information at the time of the request and there is no going back once the request is made. Information that updates from one tick to the next, such as the current location of a hat, may be asked again at the next tick. However, information that is fixed in time, such as when or where a meeting took place, can be requested only once. This model allows us to avoid, for now, the thorny issue of how to "noise up" multiple requests for the same information.

Using the payment function described in the previous section, a payment amount is mapped to a "base" probability *p*. With probability *p*, the information requested is returned in its entirety; with probability 1-*p* it is subject to noise.

There are eight basic information requests that may be made: the hats at the location, if any, participants in a meeting, capabilities carried by a hat, capability trades, meeting times, hat death time, meeting locations and hat locations. The first five requests return lists of things, such as hats, capabilities, times, etc. The latter three return single elements. How we add noise to responses to these requests depends on the type of thing requested as well as whether they involve single elements or lists of elements.

For locations and times, adding noise is treated as sampling from a normal distribution, where the mean is the location or time of the requested item, and the variance is a function of the size of the Hats world (for locations) or the amount of time since beginning the simulation (for times). Tests ensure that noisy locations are not off the Hats world map and that noisy times are not reported as having happened in the future. Adding noise to reports about a hat or capability requires sampling from the original set of hat and capability ids defined for the scenario.

Information about lists of elements is made noisy in two stages. First, the list itself is modified by discarding or adding elements. Then, with probability 1-*p*, each element of the resulting list is replaced by an element sampled uniformly from the relevant domain (e.g., replacing a true hat id by one selected at random from among all hat ids).

Information that is requested about events or entities that do not exist are also subject to noise. If noise is not applied, then a query accurately responds that the requested information does not exist. If, however, the answer is to be made noisy, then random information of the same type requested is returned.

3.3 Exporting Data

The Hats Simulator and Information Broker are designed to provide an online data feed and allow for interaction between the analyst and simulation. However, we also have implemented facilities to export batch data from the Information Broker. Hats data can be exported as perfect information (ground truth) or noisy data sets. Application of noise works differently because there is no analog of online requests with payment levels. Noise is applied to exported data in three ways: exclusion of perfect information, inclusion of false information and corruption of perfect information. The level and type of noise is parameterized. Exported Hats data has been used in several projects, including an EAGLE Program mini-TIE and controlled experiments with social network analysis tools.

4. Actions

In addition to requesting information, the analyst playing the Hats game can also change a beacon's alert level and arrest hats. Both actions affect an analyst's performance score (discussed in Section 5).

4.1 Raising Alerts

We may not be able to stop an attack, but if we know it is coming, we can prepare and minimize loss. This is the inspiration behind modeling alerts. Each beacon can be in one of three alert levels: off (default), low or high. These correspond to the conditions of no threat, a chance of an attack, and attack likely. The analyst decides which the level of each beacon alert, but the Hats Simulator keeps track of alert states over time and whether an actual attack occurs while the state is elevated. The simulator keeps statistics including counts of hits (occurrences of attacks during elevated alerts) and false positives (elevated alerts that begin and end with no beacon attack occurring). The goal of the analyst is to minimize the time beacon alerts are elevated. High alerts are more costly than low ones. On the other hand, if an attack does occur on a beacon, a high alert is better than a low alert, and a low alert is better than none.

Clayton Morrison 4/1/05 2:08 PM
Deleted: terrorist

4.2 Arresting Hats

Analysts can also issue arrest warrants for hats in order to prevent beacon attacks. Arrests are successful only when the targeted hat is currently a member of a terrorist task force. Attempted arrests under any other conditions, including hats that are terrorists but not currently part of a terrorist task force, result in a *false arrest* (a false positive). Under this model, a hat can be a terrorist but not be guilty of any crime. Unless terrorist hats are engaged in *ongoing* terrorist activities, their arrest incurs penalties. While this is a simple model, it places realistic constraints on the analyst's choice of actions.

Successful arrests do not guarantee saving beacons. A beacon is only attacked when some subset of members from a terrorist task force successfully carry the capabilities matching the target beacon's vulnerabilities to a final meeting at on that beacon. It is possible to successfully arrest a terrorist task force member but the other terrorist taskforce members still have the capabilities required to attack the beacon. However, if the analyst successfully arrests a terrorist task force member carrying required capabilities that no other task force member has, then the final meeting of the task force will take place but it will not be attacked. This is counted as a beacon *save*.

5. Scoring Analyst Performance

The Hats Simulator and Information Broker together provide an environment for testing analyst tools. The object of the game is to identify terrorist task forces before they attack beacons. Three kinds of costs are accrued:

1 The cost of acquiring and processing information about a hat. This is the "government in the bedroom" or intrusiveness cost.
2 The cost of falsely arresting benign hats.
3 The cost of harm done by terrorists.

The skill of analysts and the value of analysis tools can be measured in terms of these costs, and the Hats environment tracks them automatically as analysts play. At the end of a game, a final report is generated that includes the following four categories:

1 Costs: the total amount of "algorithmic dollars" spent on information.
2 Beacon Attacks: including the total number of attacks that succeeded and the total number of attacks that were stopped by successful arrests
3 Arrests: the number of successful arrests and the number of false arrests (false positives)
4 Beacon Alerts: the number of low and high alert hits and false positives.

6. Conclusion

Intelligence analysts tell us that Hats has many attributes of "the real thing." Some say in the same breath that Hats ought to have other attributes, for instance, telephone communications, rapid transportation of hats around the board, different kinds of beacons, and so on. We resist these efforts to make Hats more "realistic" because for us, the purpose of Hats is to provide an enormously difficult detection problem without the overhead of building rich (and probably classified) models of real domains. No doubt Hats will change over time, but we will strive to keep it simple. The other goal that guides our development of Hats is what we might call the "missing science" of intelligence analysis. To the best of our knowledge, in the current climate, analysts penalize misses more than false positives. This sort of utility function has consequences – raised national alert levels, lines at airports, and so on. Hats is intended to be a simulated world in which analysts can experiment with different utility functions. It is a laboratory in which scientific models of intelligence gathering, filtering, and use – models based on utility and information theory – can be tested and compared.

To meet these goals, we will continue development of Hats along these lines: (1) increasing the scale and efficiency of the simulator to accommodate hundreds of thousands of hats running in reasonable time to conduct experiments and play in real-time; (2) building WebHats, a web-based interface to Hats, enabling any researcher with access to the web to make immediate use of Hats as a data source; (3) providing league tables of analyst/tool performance scores from playing the Hats game, promoting public competition to better intelligence analysis technology; and (4) developing a user-friendly interface to Hats, including more complex information querying and visual aids so that human analysts can play the Hats game more naturally.

Acknowledgments

Paul Cohen and Niall Admas conceived of the Hats Simulator at Imperial College in the summer of 2002. Professor Cohen implemented the first version of Hats. Bob Schrag at IET contributed useful ideas and built a simulator similar to Hats for use in the DARPA EELD and AFRL EAGLE program. Work on this project was funded by the Air Force Research Laboratory, account number 53-4540-0588.